Distributed Systems: A Comprehensive Survey

Uwe M. Borghoff and Kristof Nast-Kolb

Institut für Informatik, Technische Universität München
Postfach 20 24 20, D–8000 München 2, West Germany
e-mail: borghoff@lan.informatik.tu-muenchen.dbp.de

Abstract

This paper gives a survey of all common transparent distributed systems. We distinguish between Distributed File Systems (DFS) and Distributed Operating Systems (DOS). Our overview is focussed on systems providing at least access or location transparency.

The paper is organized as follows: The introduction offers definitions of the features of each transparent distributed system as well as the services it is able to provide. We also propose a catalog of criteria that enables us to compare different systems independently of implementation done. The main entries we make are heterogeneity of the system's environment, communication strategy, as well as naming and security issues. Finally, we examine the reliability and availability of the separate systems and the way these issues are achieved.

The following section consists of the survey. The description of each system is organized as follows: First, we introduce the main goal the system was developed for, the classification of the system, and the provided transparency levels. Second, we try to give a short view of the advantages and disadvantages of the system. This allows our reader to compare the systems with respect to trade-off issues, such as performance versus availability. Third, a description of the system is given according to our catalog of criteria. Finally, we append information on other issues, the current status of the research, whom to contact, and a long list of references.

The next section contains a table of comparison. This table gives a lucid legible overview summarizing the information provided before and allowing for quick access.

The last section was introduced for the sake of completeness. It contains brief descriptions of related systems.

Contents

1 Introduction

This paper is yet another survey of distributed systems. However, we believe that our approach is novel. For one thing, we have tried, to the best of our knowledge, to include all distributed operating and file systems (presently in existence). For another, our approach is encyclopedic. Each system is described independently. A description of both the main goal and the pros and cons ((dis-)advantages) of each system enables the reader to decide whether or not a system is of interest to him/her.
Finally, a table comparing all systems presented was added as well as a long list of references. Nobody needs to read through the whole paper if he/she is only interested in some of the points.

1.1 Presentation of a catalog of criteria

The main goal of distributed file systems (DFS) or distributed operating systems (DOS) is to provide some level of transparency to the users of a computer network.
We have tried to develop a scheme — referred to as a *catalog of criteria* — that allows us to describe the systems in an implementation independent way. The main questions to be answered are: What kind of transparency levels are provided, how is each kind of transparency achieved, what kind of communication strategy has been proposed and finally, does the distributed character of the system allow increased availability and reliability. The last question leads us to an analysis of replication schemes used and to an evaluation of proposed failure handling/recovery strategies.

Transparency levels

We distinguish five levels of transparency. We speak of *location* transparency existing, when a process requesting a particular network resource does not necessarily know where the resource is located. *Access* transparency gives a user access to both local and remote located resources in the same way .
For reasons of availability, resources are sometimes replicated. If a user does not know whether a resource has been replicated or not, *replication* transparency exists.
The problem of synchronization is well-known. In a distributed environment this problem arises in an extended form. Encapsulating concurrency control inside a proposed system is what is meant by *concurrency* transparency. This includes schemes that provide system-wide consistency as well as weaker schemes in which user interaction can be necessary to recreate consistency. The distinction between transaction strategies and pure semaphore-based techniques is introduced in a special evaluation.
The last level of transparency is *failure* transparency. Network link failures or node crashes are always present within a network of independent nodes. Systems that provide stable storage, failure recovery and some state information are said to be failure transparent.

Heterogeneity

This survey furthermore gives information on the *heterogeneity* of the systems, i. e., assumptions made about the hardware and which operating system is used and whether that O.S. is a modified or look-alike version of another O.S.
We describe the underlying network. Is it a LAN, if so, what kind of LAN, or have gateways been developed to integrate the systems into a WAN world.

Changes made

There are two main forms of implementing distribution. First, a *new layer* can be inserted on top of an existing operating system that handles requests and provides remote access as well as some of the transparency levels. Second, the distributed system can be implemented as a *new kernel* that runs on every node. This differentiation is a first hint of how portable or compatible a system is. Some systems do not distribute all kernel facilities to all nodes. Dedicated servers can be introduced (strict client/server model). Some systems distribute a small kernel to all nodes and the rest of the utilities to special nodes (non-strict client/server model). Another group of systems

are the so called *integrated* systems. In an integrated system each node can be a client, a server or both. This survey tries to describe these differences.

Communication protocols

Message passing is the main form of *communication* (excepting multiprocessor systems which can use shared memory). We show which kind of protocols are used and describe specialized protocols if implemented.

Connection and RPC facility

The kind of *connection* established by the (peer) communication partners is another important criteria. We distinguish between point-to-point connections (virtual circuits), datagram-style connections, and connections based on pipes or streams. If a remote procedure call (RPC) facility is provided we add this information as well.

Semantics

The users of a distributed system are interested in the way their services are provided and what their *semantics* are. We distinguish *may-be* (which means that the system guarantees nothing), *at-least-once* semantics (retrying to fulfill a service until acknowledged, sometimes done twice or more frequently), *at-most-once* semantics (mostly achieved by duplicate detection) and *exactly-once* semantics. The last kind is achieved by making a service an atomic issue (so called all-or-nothing principle).

Naming strategy

We describe the *naming* philosophy and distinguish between object-oriented and traditional hierarchical naming conventions. Our overview includes the proposed name space itself as well as the mechanisms used to provide a system-spanning name space (e. g. mounting facilities or superroot-approaches).

Security issue

Security plays an important role within distributed systems, since the administration could possibly be decentralized and participating hosts cannot necessarily be trusted. Intruders may find it easy to penetrate a distributed environment. Therefore, sophisticated algorithms for encryption and authentication are necessary. We add four entries concerning this issue. First, encryption is used if no plain text will be exchanged over the communication media. Second, some systems make use of special hardware components to achieve security during the message transfer. Third, capabilities are provided that enable particular users access to resources in a secure and predefined way. Finally, we introduce the entry mutual authentication. This feature is provided if a sort of hand-shake mechanism is implemented that allows bilateral recognition of trustworthiness.

Availability

Distributed systems can be made highly available by replicating resources or services among the nodes of the network. Thus, individual indispositions of nodes can be masked. *(Nested) transactions* are well-suited in a computer network. Our overview covers this feature. First of all, we look at the concurrency control scheme, i. e. *availability* is introduced through the following mechanisms: synchronization scheme, (nested) transaction facility, and replication.

Failure handling

Failure handling/recovery is a very critical issue. Since some systems are designed to perform well in an academic environment and some systems are made highly reliable for commercial use, trade-off decisions must be taken into account. We add the following four entries to our catalog of criteria. Does the system provide recovery after a client or a server crash, does it support orphan detection and deletion, and finally, is there non-volatile memory called stable storage.

Process migration

Our final point of interest is *process migration*. Some object-oriented systems provide mobile objects, some traditional process-based systems support migration of processes. Sometimes, these approaches come along with load-balancing schemes to increase the system's performance. We include this issue in our survey.

1.2 Organization of the survey

Section 2 contains the Survey of all distributed systems in alphabetical order. Each system's overview is identically structured. First, we describe the *main goal* the system was developed for and add a brief description of the levels of transparency provided. In so doing, we also give some information on the character of the project, e. g. is it an academic or a commercial project.
Next, we try to describe the *advantages*, the pros and cons of the system. Reading both, the main goal and the advantages will sometimes suffice in deciding whether a particular system is found to be interesting or not.
If it is of interest, we include a *description* of the system in question containing all of the information needed to understand whether and how the entries of the catalog of criteria were achieved.
Finally, we append *miscellaneous* information, such as other issues of the system including applications or special software implemented to support the approach. We describe the current status of the project and whom to contact to get more information and, a more or less complete list of references to help the reader in his/her investigation.
Our survey ends with a Table of Comparison (section 3). Everybody knows that, in certain ways, a categorized comparison can be misleading. Therefore we recommend careful use of this table. It summarizes the main entries of our catalog of criteria and provides for every system the degree of to which it fulfills them.
Section 4 contains a list of Related Systems which aren't surveyed in section 2 because they do not provide the necessary transparency level(s) or because they are of minor interest (e. g. predecessor versions of a surveyed system). They are mentioned for the sake of completeness. A brief description and a main reference is given for each system.

Disclaimer: Due to the nature of our paper, we know that the up-to-date status can't be reached. There is a good chance, that new publications of our surveyed systems or new systems at all are not mentioned. Moreover, contact addresses may have changed.

2 The Survey

2.1 Accent

Main Goal
Accent is a successor to the RIG system (cf. section 4), a message-based network access machine. It tries to solve the problems of transparent access, failure notification, protection and access to large objects (e. g. files) with low message overhead. Also, it provides location and failure transparency.
Advantages
Accent achieves these goals efficiently by introducing capabilities for message ports and by the use of copy-on-write virtual memory management throughout the network. The disadvantage of the system is its restriction to one type of hardware and that UNIX software cannot be run on it.
Description
Accent consists of a new operating system kernel which operates only on PERQ workstations. The IPC facility is supported under a modified version of UNIX 4.2 BSD.
The inter-process communication of Accent uses ports as an abstraction to which messages could be sent asynchronously. Each process can have various ports, e.g. a kernel port. The rights for possessing a port and for sending or receiving messages are distinguished. The semantics are application dependent, but at-most-once can be achieved. If a port is full, the sending process will be suspended or will be informed of this. Rather than copying large messages all at once, they are handled copy-on-reference even across the network, making virtual memory possible. All communication over the network is done by special network servers running on each host which forward messages transparent to the senders. Each network server maintains a mapping between network ports

(ports, which are remotely accessible) and corresponding local ports. The use of these servers has the advantage that they can do data type conversions as well as the implementation of secure transmission outside the kernel.
Every access to a port is protected by the use of capabilities, given that every kernel is trusted. The network servers map local to remote capabilities. Processes are allowed to move from one machine to another transparent to other processes but all network servers must get this information.
Since ports differentiate between the owning and receiving process, it is possible to give the surviving process of a crash both rights automatically to assure further computation. Additional reliability issues must be implemented on top of the given system.

Miscellaneous

Status: Accent is completed and used on 150 PERQ workstations at Carnegie Mellon University. It leads to the evolution of Mach.
Contact: Richard F. Rashid, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
References: [1], [2], [3], [4], [5], [6], [7]

2.2 Alpine

Main Goal

The primary purpose of Alpine (known within Xerox as *Research Alpine* or *rAlpine*) is to store database files. Its secondary goal is to provide transparent access to ordinary files, such as documents, programs, and the like.

Advantages

The database system at Xerox runs on top of the Alpine transaction-based file system. The clear disadvantage of file-level transactions is low performance since a file system sets locks on files or pages. In contrast, a database system is able to perform its own concurrency control and to set locks on logical units. Alpine gives the client no way to find out why a transaction has aborted.
On the other hand, an advantage of the above philosophy of file-level transaction is that it provides the option of running the database system on a separate machine from the file system.

Description

Alpine is written in the Cedar language (cf. section 2.9), using the Cedar programming environment. Alpine runs on Dorado workstations.
All communication is performed via a (Cedar) general-purpose Remote Procedure Call facility. A compiler, called *Lupine*, takes a Cedar interface as input and produces server and client stub modules as output. The client stub module exports the Cedar interface access transparently, translating local calls into a suitable sequence of packets addressed to the server.
A client must call the RPC system to establish a *conversation* before it can make authenticated calls. The conversation embodies the authenticated identities of both the client and the server, and the security level (encrypted or not) of their communication.
A file is named by a file-Id that is unique within a so called *volume* containing the file. A volume is an abstraction of a physical disk volume. Volumes are named and have globally unique Ids.
Alpine uses a *(redo) log*-based technique to implement atomic file update, and performs the standard two-phase commit protocol. The locking hierarchy has two levels: entire-file and page locks. A client chooses between these two levels when opening a file. The log is represented as a fixed-length file in the underlying file system, divided into variable length log records, allocated in a queue-like fashion. It implements a simple access control scheme with a read and modify access list for each file. The Alpine file system supports atomic transactions. Alpine treats a *transaction-Id* as a capability. Any client that has a transaction's Id and the name of the transaction's coordinator may participate in this particular transaction. Clients participating in the same transaction are responsible for synchronization since locks do not synchronize concurrent procedure calls for the same transaction.

A *lock manager* detects deadlocks within a single file system by timeouts and aborts transactions to break these deadlocks.
Replication of files is not supported.

Miscellaneous

Other issues: Xerox conceived of building Alpine in December 1980. Three external components have been built to make Alpine usable from Cedar: An IFS-like hierarchical directory system, an implementation of Cedar open file, and a user interface to the server's administrative functions.
Status: By early 1983 the implementation was coded and started to test. Xerox decided to replace IFS (cf. section 4) and XDFS (cf. section 2.52) by Alpine.
It is planned to include a distributed directory system with location transparency and some form of file replication.
Contact: M.R. Brown and K.N. Kolling, Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, CA 94301
or E.A. Taft. Adobe Systems Inc., 1870 Embarcadero Rd., Suite 100, Palo Alto, CA 94303.
References: [8], [9]

2.3 Amoeba

Main Goal

The main goal of Amoeba was to develop a capability-based, object-oriented distributed operating system, which is reliable and performs well. It provides access-, location-, concurrency- and failure-transparency.

Advantages

The main advantages are its fast inter process communication and the possibility to implement security issues for the use in open systems.
The disadvantage is that extra time and a special directory server are needed to convert ascii-names into capabilities.

Description

To run Amoeba, a special small kernel has to be installed. Until now implementations for the 68000-family, PDP-11, VAX, IBM-PC and NS32016 have been available.
For compatibility with UNIX an *Amoeba-interface* must be installed at the UNIX machine. UNIX file- and process-servers are required at the Amoeba site.
The communication uses a special four-layer protocol consisting of the physical layer, the port layer which locates ports and offers a datagram service, the transaction layer which transmits messages reliable and the final layer which is application dependent. The RPC provides an at-most-once semantics with the help of acknowledgements, timers and *are-you-still-alive?*-messages. If the call is idempotent *at-least-once*-semantics can be achieved, even if a server crashes, by using another server. The available message transfer primitives (send, get_request and put_reply) are implemented as small library routines. Although these primitives are blocking, special exception messages may interrupt a receiving process.
Each object has a global unique name, expressed through its capability. A capability contains the port number of the service which manages the object, the local number of the object and a field of rights. To keep the port number unique, they are random integers with 48 bits. Capabilities can be stored in directories managed by a directory service allowing hierarchical (UNIX like) directory trees. Each client has a table in order to associate physical addresses to port numbers. If a service is new or has changed its locality, a broadcast is made. To allow use of services via wide area networks, there is a special server in the local net that acts like the far distant server and forwards all requests.
Partial authentication of clients is achieved through the use of capabilities as described above. To prevent an intruder from impersonating a server, a so called *F-box* is inserted between each processor and the network. The disadvantage is that security isn't guaranteed if a person omits the F-box at his/her own workstation.
Amoeba includes several measures for improving availability and reliability:

A boot server tries to restart crashed servers. To detect orphans, the *are-you-still-alive?*-messages are used: If a server doesn't get these messages, it assumes that the client has crashed and kills the associated processes.
In the Amoeba file system (FUSS), the file- and block-servers are replicated, a stable storage server exists and updates are atomic. To preserve consistency, optimistic concurrency control is combined with locking. At each update a new version is created and only when a version is committed, concurrent updates have to be checked. If they are not serializable (read-/write-bits are set for each block) a manual intervention becomes necessary. Locks can be set for updates of groups of files in which conflicts are more likely.

Miscellaneous

Other issues: A bank account server administers resources. Status: Amoeba is being developed and further improved at the Vrije University and the CWI in Amsterdam. It is used at sites in four European countries which are connected via a wide area network.
Contact: Sape J. Mullender, CWI Amsterdam, Netherlands
or Andy Tanenbaum, Dept. of Mathematics and Computer Science, Vrije Universiteit, Postbus 7161, 1007 MC Amsterdam, Netherlands.
References: [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32]

2.4 Andrew

Main Goal

Andrew is a distributed computing environment being developed at the Carnegie Mellon University, Pittsburg in a workstation/server principle.
The goals of the Andrew file system are to support growth up to at least 7000 workstations in a campus-wide connected network while providing students, faculty members and staff with the amenities of a shared file system. Using a set of dedicated servers, collectively called *Vice*, the Andrew virtual single file system presents a homogeneous, location-transparent file name space to all of its client workstations.

Advantages

Andrew shows that connecting a moderate amount of heterogeneous workstations is tractable. But there are several disadvantages: Caching is the main form of replication, resulting in a low degree of availability. Performance is poor due to the bottleneck of the server CPUs. Obviously this criticism is focused on Andrew itself. The performance of a multi-workstation DFS providing replication cannot be compared with a small size DFS that does no replication at all (e. g. NFS (cf. section 2.39) or RFS (cf. section 2.43)).

Description

Clients and servers run on top of a modified UNIX 4.2 BSD kernel. Supported machine types are SUN 2/3 workstations, DEC MicroVAXes and IBM RT PCs. Typical Andrew workstations applications (*Virtues*) use the campus IBM Token Ring Network to communicate with the *Vice* file system. *Vice* is composed of a collection of semi-autonomous clusters connected together by a backbone LAN. Each cluster consists of a collection of *Virtues* and a representative of *Vice*, the so called *Cluster Server*. The operating system on each workstation intercepts file system calls and forwards them to a user-level process called *VENUS*.
Vice and *Virtue*, based on a client/server model, communicate by a RPC mechanism using the Unreliable Datagram Protocol supported by UNIX.
From the point of view of application programs the name space of file names is partitioned into two subspaces: local and shared. Both are hierarchically structured. The local name space is the root file system on the workstation and the shared part is a mounted file system branch from elsewhere. In practice almost all files accessible to users are in the shared space. So called *crosscluster* accesses are costly and time consuming and should be reduced if possible.
On demand, entire files are cached on local disk. Important files can be replicated at all the cluster servers. There

is no block paging over the network. Therefore diskless workstations are not supported. A so called *volume* has been introduced. A volume is a subtree of the shared files. Each user has his/her own volume attached to him. The necessary file location information is obtained from a volume location database. This database is replicated at all servers.
Security within Andrew: *Vice* is the boundary of trustworthiness. Everything within *Vice* is assumed to be secure. *Virtue* is under user control and is never trusted by *Vice*. After mutual authentication *Vice* and *Virtue* communicate only via encrypted messages with a key generated at the beginning of each session. These features are integrated into the remote procedure call package. *Vice* obtains access lists to protect shared files. These lists are only associated with directories. Files within a directory may be individually protected but selective access rights to different users are not possible. The user rights on a shared file are the union of all the rights specified for all the groups to which the user belongs.
Vice provides multi-readers single-writer synchronization. Application programs do not have to do explicit locking, which is implicitly done by the caching software.
Availability: Single point network or machine failures are accepted resulting in a temporary loss of service to a small group of users. A check-on-open file exists to reduce server state information. This means that after a file close the cache copy is transmitted to its appropriate location. Changes are immediately visible to all other users as in a timesharing file system.

Miscellaneous

Other Issues: Remote program execution. The so called *Butler System* composed of a set of programs running on Andrew workstations that give users access to idle workstations.
Status: Andrew runs campus-wide and has approximately 350 *Virtue*s on line with about 3600 registered users.
Contact: Mahadev Satyanarayanan, Information Technology Center, Carnegie-Mellon University, Schenley Park, Pittsburg, PA 15213
References: [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51]

2.5 Argus

Main Goal

Argus is both a programming language for writing distributed applications as well as a system developed to support the implementation and execution of distributed environments. It copes with the problems that arise in distributed programs, such as network partitions and crashes of remote sites. On the other hand Argus does not free a programmer from the concerns of concurrency: Deadlocks and starvation are not detected or avoided. Argus is transparent in the sense of access, location and failure.

Advantages

Argus provides two novel linguistic mechanisms: First, *guardians* implemented as a number of procedures that are run in response to remote requests. A *guardian* survives failures of the node at which it is running (resilient). A *guardian* encapsulates a resource that is accessed via special procedures called handlers. Second, *atomic transactions*, or *actions* for short, can be nested and mask problems arising from failures and concurrency.
The main disadvantage of Argus is its poor performance. Two reasons: First, Argus treats each call as a complete transaction. Second, the Argus implementation resides on top of an existing operating system.

Description

The Argus implementation runs on MicroVAX-IIs under Ultrix 1.2 connected by a 10 megabit/second Ethernet. It is a new layer on top of Ultrix. No kernel modifications are made except for an optimized UDP, the so called ACP (Argus Communication Protocol). Argus uses message-based communication (datagram service), arguments are passed by value. ACP has 32-bit port numbers, *guardians* have system-wide 32-bit unique identifiers. Individual packets are tagged with a message unique identifier and a sequence number. The final package indicates the end of a message.

Argus provides built-in types of atomic objects, such as atomic arrays or atomic records and allows a programmer to define his/her own atomic data types.
Semantics: Every handler call is a subaction and happens exactly-once.
Naming in Argus: To send a message, a so called *thread* first has to map the destination *guardian* identifier to a network address.
Security issues play no role.
Availability and reliability: A strict two-phase commit protocol supports the atomicity of transactions. Serializability is implemented via atomic objects providing a set of operations to access and manipulate these atomic objects. Recovery is done using versions written periodically to stable storage. The hierarchical structure of actions (action tree) allows orphan detection by using special state-lists of action participants (orphan destruction algorithm).

Miscellaneous

Argus is also a programming language which gives a programmer the possibility to deal with distributed applications (extension of the CLU language). Argus is unique because it provides atomic actions within a programming language.
Argus has been used to implement a number of distributed programs, such as a library information system, a distributed editor, and a mail repository.
Status: The Argus research is active.
Contact: Barbara Liskov, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, MA 02139
References: [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64]

2.6 Athena

Main Goal

Project Athena is an educational experiment introduced in May 1983. It is a joint project of MIT, DEC and IBM to explore the potential uses of advanced computer technology in the university curriculum. Athena provides some sort of location and access transparency through dedicated service machines.

Advantages

Athena is a novel approach because it is the first attempt to use and integrate the computer power of a large campus-wide network into an academic curriculum.

Description

MIT has installed a network of about 2000 high-performance graphic workstations. The network is implemented with multi-vendor technologies (such as the redundant Proteon Ring) and is based on a high-speed backbone network. Existing campus facilities have access to LANs connected to this backbone network via special purpose gateways.
Athena uses a single operating system and a single communication protocol family to implement its computing environment. The operating system is UNIX, the protocol family is TCP/IP. This transport protocol is used to design the same virtual network for all machines in the project.
User-id and login names are unique across all Athena nodes. The name space of files is UNIX-like and hierarchical. Computers connected to the network can be divided into two categories: *service computers* and *workstations*. *Service computers* provide generic services such as filing or authentication to one or more client computers. It is possible to share information and to access data and programs from any computer. Servers provide this access to system libraries, to class-specific libraries, and to user lockers. A user locker contains all user-specific resources.
Replication, failure transparency and network-wide location transparency are not provided.

Miscellaneous

Status: Athena is still in use for student education. A network-wide file system is planned.

Contact: Edward Balkovich, Digital Equipment Corporation, 200 Forest Street, MR01-1/L26, Box 1001, Marlboro, MA 01752-9101
or Steve Lerman, Massachusetts Institute of Technology (MIT), Cambridge, MA 02139.
References: [65], [66], [67], [68]

2.7 BirliX

Main Goal

BirliX is an object-oriented operating system that supports distributed, reliable, secure and parallel applications by providing suitable services. A major point of the architecture is failure detection and recovery. All levels of transparency are achieved.

Advantages

The system tries to simplify the resource access, compared with UNIX, by the unified use of objects. Most possible failures of a distributed systems are handled. Due to the early state of the project it is difficult to judge. However, it seems not to scale very well because a large amount of data has to be kept on each site.

Description

BirliX emulates the UNIX 4.3 BSD interface providing 90 percent object code level compatibility.

Within *teams* communication via shared memory is possible and synchronized by monitors. Teams communicate synchronously by messages which have a fixed small length or are of arbitrary length. Messages that are passed across the network are handled by assisting processes on both sides so that the kernel is kept small. Error free message passing isn't provided because this can be done more efficiently inside the teams. Supported are timeouts, sequence numbering to detect duplicates at existing client/server relations and searching for previous relations at the start of a new one.

There are four fundamental abstractions: objects, teams for implementing objects and segments and processes for physical resources. A three-level naming scheme is used. First an object is searched by a network transparent, hierarchical name in a given context. The associated unique identifier is searched for in a data base which is replicated at each node. The corresponding team and the agent serving this client can be found with this unique Id. The whole network is searched for the object provided the local replica of the data base contains no entry. This is possible because the data base can be inconsistent. For performance improvements any location hints are cached.

Access to objects is checked by the object manager, which forms a capability afterwards. Users are authenticated by passwords. To protect capabilities from forgery either reliable sender identification or encryption algorithms should be used. The name space is divided into two main parts: */common* for all files on servers and */station/stationname* for the specified workstation in order to protect private name spaces.

To recover from failures, objects and processes can be checkpointed in any state. Application programs form so called *recovery clusters* that are created as one unit with regard to recovery. In the case of station failures all teams on non-failed stations belonging to this cluster are terminated. Then all teams are recreated from their checkpoints where each segment of memory has to be mapped to the same range of addresses. In addition a garbage collection is executed. A central mechanism exists to replicate objects. The checkpointing of replicas is synchronized.

Miscellaneous

Other issues: Objects can migrate to another host by deleting the team and creating it at another site.

Status: A prototype of the system has been running since the beginning of 1988. Current work is concerned with the implementation of a new prototype which includes all proposed features, improved performance and simplified internal interfaces.

Contact: Hermann Härtig, Gesellschaft für Mathematik und Datenverarbeitung mbH, Schloss Birlinghoven, 5205 St. Augustin 1, West Germany
References: [69], [70], [71], [72], [73]

2.8 Cambridge DS

Main Goal

The main intention at Cambridge was to use the Cambridge Digital Communication Ring by a coherent system. The Cambridge Distributed Computing System is primarily composed of a processor bank and some servers. It provides access, location and failure transparency.

Advantages

The Cambridge DS was one of the first distributed systems. Despite this fact it provides good performance and reliable services. A disadvantage is that it is hardly portable because it is tailored to a very specific hardware, the Cambridge Ring.

Description

The processor bank consists of LSI-4 and Motorola 68000 based machines. Most of the servers are running on Z-80-based computers or on a PDP 11/45. Each user has an associated TRIPOS single-user operating system.
Nearly all communication is built on sending packets of up to 2kBytes which consists of 2Byte minipackets which can be transferred on the Cambridge Ring. On top of this are a simple remote procedure call protocol and a byte stream protocol. The RPC consists only of requests and responses without acknowledgement (may-be semantics). The byte stream protocol is full-duplex and connection oriented with flow and error control.
Services can be located by using a name server which returns the machine number, the port and the protocol it expects. For location of files a *filing machine* lies between the file server and the accessing machine. This machine does name lookup and caching to improve performance. Files of the user's TRIPOS file system are mapped automatically by stubs to files of the file server.
The protection system is based on the use of capabilities. At each access, a capability is sent to the *Active Name Server* where it is verified. User authentication is made at login with name and password. Byte streams can also be authenticated.
The file server distinguishes normal and special files which are updated atomically. A garbage collector program checks the file system for inaccessible files which are deleted. Availability of the system is improved by the use of a processor bank. The main approach for fault tolerance is to bring up servers after a crash by detecting the crash and downloading of the appropriate program.

Miscellaneous

Status: The project is completed and is in use at the Computer Laboratory at Cambridge.
Contact: Roger Needham, Computer Laboratory, University of Cambridge, Cambridge CB2 3QG, England
References: [74], [75], [76], [77], [78], [79], [80], [81], [82], [83]

2.9 Cedar

Main Goal

The Cedar File System (CFS) was developed as part of the Cedar experimental programming environment at the Xerox Palo Alto Research Center. CFS is a workstation file system that provides access to both a workstation's local disk and to remote file servers via a single hierarchical name space. Sharing of highly available and consistent file versions was the main goal of this project.

Advantages

The combination of the Cedar File System's semantics together with the tool for maintaining consistent versions of shared files is an interesting approach for a highly available and consistency-guaranteeing distributed file system. The main disadvantage of this system is implied by the concept of immutability of files. Each version of a file is a complete copy resulting in an enormous amount of disk space and an increased disk allocation activity. There is no attempt made to reduce storage space by differentiating subsequent versions or by deleting unneeded versions.

Description

The CFS is implemented by Cedar workstations (Alto and Dorado workstations) and a collection of the *Interim File System servers*. CFS accesses IFS servers through an inter-network using the File Transfer Protocol. The CFS is fully implemented inside the workstations code. CFS is designed to support sharing of file server remote files by workstations connected via a LAN. Each workstation is equipped with its own local disk. Diskless stations are not supported. CFS provides each workstation with a hierarchical name space that includes the files on the local disk and on all file servers. Local files are private, remote files are sharable among all clients. To make a local file available to other clients, the file is transferred to a file server by giving it a remote name. Other workstation clients are now enabled to access this file by its remote name and to transfer it to their own local disk. Remote file creation is atomic. Only entire files are transferable.

It is now easy to understand why CFS only provides access to shared files which are immutable. An *immutable file* is a file that cannot be modified once it has been created. This leads to the naming strategy in CFS. Updating of shared files is accomplished by creating new file versions. For this reason, a complete file name consists of a server name, a root directory, some subdirectories (not necessarily), a simple name and a version. The server part identifies the server at which the file is allocated. Two variables provide access to the lowest (*!low*) and the highest (*!high*) versions of a given file.

Even local files are treated by compilers and editors as immutable by always creating new file versions when results are written back to disk.

For security reasons, CFS uses the access control system provided by IFS for user identification. Access control lists contain user access rights. The information about a user — an user's name and his password — is passed to IFS as part of the FTP.

Availability plays an important role in CFS and is achieved by replication. Important directories are automatically as well as periodically replicated to multiple servers by a daemon process. All read requests to open remote files are satisfied from a local cache. For this purpose, every workstation reserves a portion of local disk space as cache. The cache is managed using an approximate LRU (least recently used) strategy.

Consistency of files is achieved by the concept of immutability of files — creating a new file version every time a particular file is modified — , consistency of file server directories is achieved by performing a transaction for directory updates. Updates are indivisible and serialized. Thus, transferring a new file to a server, assigning it a new version number, and entering its name in the file server directory appear to be a single act.

Miscellaneous

Other issues: An application of the Cedar File System is the support of the so called *Cedar Software Management Tools* known as the *DF system*. DF provides an environment for the handling and maintaining of large software packages. A programmer using some DF files can be sure to have a consistent view over these subsystem files.

Cedar is also a programming language. The Alpine File System (see section 2.2) has been written in the Cedar language.

Status: Research on CFS is active and continues to be enhanced. CFS is still in use as an integral part of the Cedar Programming Environment.

Contact: Michael D. Schroeder, DEC Systems Research Center, 130 Lytton Ave., Palo Alto, CA 94301

References: [84], [85], [86], [87], [88], [89], [90], [91], [92]

2.10 Charlotte

Main Goal

Charlotte is a distributed operating system developed at the Computer Sciences Department of the University of Wisconsin–Madison providing location and access transparency. It is part of the Crystal project that was funded starting in 1981 to construct a multicomputer with a large number of substantial processing nodes.

Advantages

The Charlotte kernel provides multiprocessing, mechanisms for scheduling, storage allocation, and process migration. The main design decision for the Charlotte operating system is to keep the kernel small, efficient, concise, and easily implemented. Therefore, only those services are included in the kernel (such as IPC and process control) which are essential to the entire system. All other services are implemented as utility processes outside the kernel. The main disadvantage of Charlotte is the file server design. A weak protection scheme and an unusual directory handling service make it rather difficult to use in practice.

Description

Charlotte is implemented as a new kernel which runs on about 40 VAX–11/750 and is connected by a Proteon ProNet local area network with a bandwidth of up to 80 Mbit/second. It is written in a local extension to Modula except for the startup code which is written in assembler and the interprocess communication code which is written in C.

A software package called the *nugget* resides on every node. Nugget enforces allocation of the network among different applications by virtualizing communications, disks, and terminals.

Charlotte provides a unique form of IPC. A *link* is a capability-protected connection between two processes. Processes address each other by presenting these capabilities (*link-Ids*). Information about the location of a process is hidden inside the kernel. The kernel maintains a link table to locate a particular link. The process-level communication is non-blocking (asynchronous) as well as synchronous and unbuffered. Unbuffered means that a message is not transmitted until the receiver has provided enough space to handle it. The IPC scheme described above is implemented by means of a finite-state automaton for each link.

Processes under Charlotte do not share memory.

Any client can gain access to only a subset of the files. If a request is outside of its jurisdiction, a forward to the appropriate remote file server is initiated. The link to the client is part of the forwarding message.

In Charlotte, each file has an unique full path name. Protection is implemented on a file-by-file basis, not by evaluating the rights in directories on a particular path. Therefore, to delete a file, it must be removed from the server on which it resides as well as from the directory. Renaming of directories is not supported.

Failure handling in Charlotte is an optimistic approach. If a machine fails, it is assumed that within a small amount of time all kernels will discover this and send a *link destroyed* message to all local clients.

Replication of resources is not supported. As mentioned above, process migration is part of the design decisions.

Miscellaneous

Status: The utilities of Charlotte which have been designed and prototypically implemented so far include a switchboard, a program starter to manage the memory, and the file server.

Contact: Raphael Finkel or Marvin Solomon, Computer Sciences Department of the University of Wisconsin–Madison, Wisconsin, USA.

References: [93]

2.11 Chorus

Main Goal

The Chorus project was launched at INRIA at the end of 1979 with the objective to study the basic concepts of supporting distributed systems. Chorus is intended to provide the basis upon which the various tools for distributed computing could be installed efficiently and with as little effort as possible. Chorus provides access and location transparency.

Advantages

Chorus is a small kernel on which a number of different operating systems can run simultaneous. It tries to specify layered communication protocols unambiguously. This approach — in the early 80s — was in line with the work in progress within ISO or CCITT.

Description

Chorus (Version 3) runs on different computer architectures based on MC 68000s and Intel 386 (e.g. IBM PC/AT and Bull SM 90) interconnected through an Ethernet local area network. It provides an extended UNIX System V interface which can be used without changes by existing UNIX applications.

Chorus is a new message passing kernel which structures distributed processing into *actors*. Actors are local sequential processes. Each actor is located on a single site and can manipulate local objects. Actors communicate only by explicit exchange of asynchronous messages. A set of cooperating actors form an *activity* which is the distributed processing unit. Actors are mobile. The operating systems, e.g. UNIX, are implemented as servers on top of the kernel.

Actors send and receive messages through *ports*. Wether local or remote, ports are an uniform view of communication. Actors do not necessarily know where correspondent ports are located. RPCs, asynchronous message passing and multicasting are supported.

In Chorus, applications are described in terms of modules. A module is a group of procedures which are always kept together in a single actor.

Each entity (object, actor, port) is uniquely identified with a global name. This name is not used again for designating any other entity.

Protection in Chorus. Objects can be manipulated only by locally residing actors which have obtained a so called *link* to them. A *linkage control procedure* checks the validity of a special request. Objects can be protected by periodically changing passwords revealed only to authorized actors.

Replication, transaction or failure handling are not mentioned.

Miscellaneous

Other issues: Light-weighted processes, distributed virtual memory and real-time facilities are available. In addition processes can be executed remote.

Status: A prototype implementation has been constructed and tested at INRIA to support various distributed functions developed in other research projects such as *SIRIUS* (Distributed Data Bases) or *KAYAK* (Office Automation). The research effort is now transfered to commercial venture, Chorus système, with Chorus-V3 as their current version. Interfaces to UNIX BSD, OS/2 and object-oriented systems are planned.

Contact: Frédéric Herrmann, Chorus systèmes, 6 Av. Gustave Eiffel, F-78183, Saint-Quentin, en-Yvelines Cedex, France or

M. Guillemont, INRIA Domaine de Voluceau – BP 105 – 78153 Le Chesnay Cédex, France — now at Chorus Systems.

References: [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105]

2.12 Clouds

Main Goal

Clouds is a distributed operating system developed to provide the integration, reliability and structure that makes a distributed system usable. The structure of Clouds promotes location, access, concurrency and failure transparency, and support for advanced programming paradigms, and integration of resource management, as well as a fair degree of autonomy at each site.

Advantages

Clouds provides system researchers an adaptable, high-performance (in the future when version 2 is fully implemented) workbench for experimentation in distributed areas, such as distributed databases, distributed computing, and network application. One disadvantage of Clouds is that there is no way to subclass Clouds objects. Also, inheritance is not supported.

Description
The first Clouds version was built on VAX-11/750 hosts which were connected through an 10 Mbit/sec Ethernet. The current version has been implemented for the SUN 3 workstations. The cluster of Clouds machines runs over an Ethernet, and users will be able to gain access to them through workstations running Clouds as well as any other UNIX workstation.
Clouds is implemented as a new kernel (called *Ra*) which is a native kernel running on bare hardware.
Clouds is designed as an *object/thread* model. All instances of services, devices, programs and data are encapsulated in entities called *objects*. *Threads* are the only active entities in the system, and are used to execute the code in an object. Object invocation can be nested. Local and remote invocations (RPC-based) are differentiated by Clouds and are transparent at the application level. Distributed shared memory allows remote objects to be moved to the local node so that the invocation may execute locally. Threads do not need any communication through messages. Thus ports do not need to be supported.
Every object is typed and named in a flat name space.
No security issues are implemented extending UNIX.
Segment locking and synchronisation mechanisms can be used by applications to coordinate access to shared segments.
Clouds (similar to Argus, section 2.5) provides nested transactions and locking semantics. Stable objects are implemented and modified by atomic operations. The permanent copy of an atomic object is updated only when a commit is performed. A storage manager provides the commitment using the strict two-phase commit protocol.
Miscellaneous
Other issues: The language *Aeolus* has been designed to integrate the full set of features that the Clouds kernel supports.
Status: Research is going on to port Clouds to Sun 4, Sun 386i and Sequent multiprocessor architectures and to design and implement more system objects.
Contact: P. Dasgupta, School of Information and Computer Science, Georgia Institute of Technology, Atlanta GA. 30332
References: [106], [107], [108], [109], [110], [111], [112], [113], [114], [115], [116], [117], [118]

2.13 Cosmos

Main Goal
Cosmos has been designed at the University of Lancaster, U. K. Cosmos gives integral support for both distribution and programming environment functions. Location and access transparency, and eventually replication and synchronization transparency are provided.
Advantages
Cosmos is not implemented yet, just simulated. But the Cosmos approach provides a unifying framework for handling distribution and for supporting a more sophisticated programming environment.
Description
Cosmos will be implemented as a new kernel. Each node in Cosmos will run a copy of the kernel. Currently, a simulation of the Cosmos system runs under UNIX. A prototype implementation is planned to run on an Ethernet local area network.
Cosmos is one of the most recent projects that build a distributed system as an object model. An object encapsulates persistent data items and is manipulated by the use of specific operations. All accesses to data objects must be through a well-defined operational interface (similar to Eden (section 2.19) or Clouds (section 2.12)).
In Cosmos, objects are *immutable* (see Cedar, section 2.9). Once created, an object will never be changed. Instead, objects are transformed atomically leading to a graph of object versions.

To increase the availability of the system, objects may be replicated. An algorithm was developed for maintaining the mutual consistency of the replicas based on Gifford's weighted voting scheme.

Miscellaneous

Status: A simulation of the Cosmos kernel running under UNIX has been developed. Some kernel facilities are already implemented. A full implementation is being planned on a network of computers interconnected by an Ethernet LAN.

Contact: Gordon S. Blair or John R. Nicol, Department of Computing, University of Lancaster, Bailrigg Lancaster LA1 4YR, U. K.

References: [119], [120], [121], [122]

2.14 Cronus

Main Goal

The Cronus distributed operating system was developed to establish a distributed architecture for interconnected heterogeneous computers promoting and managing resource sharing among the inter-network nodes. Cronus is based on an abstract object model providing location and access transparency.

Advantages

Cronus is an approach that shows that designing a distributed operating system which operates on and integrates a set of heterogeneous architectures in an inter-network environment is tractable. The main disadvantage is that Cronus is hardly portable due to the host-dependent kernel implementation.

Description

Cronus is a new operating system kernel which operates in an inter-net environment including both geographically distributed networks with a span of up to thousands of miles, and local area networks. Cronus currently runs on a single cluster defined by one or more LANs. The principal elements of a Cronus cluster include first of all a set of hosts upon which Cronus operates. Second, some LANs which support communication among hosts within a cluster and finally, an internet gateway which connects clusters to a large internet environment.

The current Cronus development uses Ethernet and Pronet Ring LANs. Local network capabilities are indirectly accessed through an interface called *Virtual Local Network* (VLN). It is based on a network-independent IP datagram standard augmented with broadcast and multicast features. The LANs are connected via an ARPA standard gateway (TCP,UDP/IP).

The testbed configuration consists of BBN C70 under UNIX V7, DEC VAXes under VMS and UNIX 4.2 BSD, SUNs under UNIX 4.2 BSD, and GCE 68000 systems.

Cronus is designed as an abstract object model. All system activities are operations on objects organized in classes called *types*. The Cronus IPC is cast in the form of operation invocation and replies, both synchronously and asynchronously. Communication is a superset of the RPC scheme.

Each object is under direct control of a manager process on some host in the Cronus cluster.

Associated with all Cronus objects is a unique identifier which is a fixed-length structured bit string guaranteed to be unique over the lifetime of the system.

User identity objects, called *principals* (capabilities), are used to support user authentication and the access control mechanisms. The principals are managed by an authentication manager. In a typical login-sequence the user supplies a name of a principal and a password for that principal object.

Some objects can migrate to serve as the basis for reconfiguring the system. Other objects are replicated to increase the availability of the system.

Miscellaneous

Other issues: The *Diamond multi-media message system* is an application running under the Cronus operating system.

Status: Active. An additional cluster is planned for installation at the Rome Air Development Center.

Contact: R.E. Schantz or R.F. Gurwitz, BBN Laboratories Incorporated, 10 Moulton Street, Cambridge, MA., 02238.
References: [123], [124], [125]

2.15 DACNOS

Main Goal
The Distributed Academic Computing Networking Operating System is part of HECTOR, a joint project of IBM Germany and the University of Karlsruhe, W-Germany. The main design objectives are preservation of investment, that means that DACNOS must appear as a compatible add-on to existing systems, node autonomy, convenience to the user, i. e. transparent access to remote resources as well as openness and portability.
Advantages
DACNOS is one of the few attempts to provide transparency on top of heterogeneous operating systems. The DACNOS prototype has been operational for only a short time, so that not much can be said about performance or feasibility of the system in an academic environment.
Description
The prototype has been implemented under three types of operating systems: PC DOS on IBM PC AT, VM/CMS on IBM S/370 and VMS on DEC VAX.
A Global Transport component (GT) interconnects multiple vendor network architectures, e.g. SNA and provides a reliable transport service, which conforms to the OSI layer 4 service interface. The GT transfers datagrams that are reliable by establishing a connection to the target transparent to the user. The *Remote Service Call* (RSC) operations provide an RPC-like mechanism. The operations are available as calls in a programming language. In addition, if necessary, they perform conversion of data.
Global unique names are achieved by concatenating hierarchical domain names and local names within these domains. The *Distributed Directory System* administers these names and locates remote objects by asking a subset of servers which, if necessary, subsequently forwards the query.
The *Remote File Access* (RFA) component of DACNOS defines a unified, homogeneous file system on top of the heterogeneous local file systems. The global file names are sequences of identifiers separated by periods. Files become visible in the network by publishing. These files (or file sets) must be mounted and have to be attached to a virtual local file name to guarantee transparent access.
Each user has to be authenticated by the authentication server of his home domain. A server keeps access control lists for all its objects. If a user requests an operation on an object, the server checks the authorization with the help of the server in the user's domain. For this purpose, one authentication server must exist in each domain, installed under a fixed logical node name.
Concurrent access to files is synchronized in such a way that a writer gets a new version which replaces the original file as soon as the writer closes the file. In the meantime multiple readers may read the old version.
Miscellaneous
Other issues: The *Remote Execution Service* selects execution sites for running programs remote, can control the running program and provides access to resources that belong to the environment of the originating unit.
Status: The project has been completed since April 1988, and a prototype has been installed. An evaluation of DACNOS is planned in a realistic environment.
Contact: H. Schmutz, IBM European Networking Center, Postfach 103068, 6900 Heidelberg, West Germany or O. Drobnik, University of Karlsruhe, Postfach 6980, 7500 Karlsruhe, West Germany
References: [126], [127], [128], [129], [130], [131], [132], [133], [134], [135], [136], [137], [138], [139], [140], [141]

2.16 DEMOS/MP

Main Goal

The main goal of the DEMOS/MP project was to provide a basis for various experiments in distributed systems within an easyly modified and well structured system. Access and location transparency should be provided and failures of nodes and the network should be handled transparently.

Advantages

The system is simply organized and structured. The two common forms of communication, short messages and bulk data transfer, are well supported. The system tries to improve reliability by a recovery mechanism, but this mechanism and the global name service are centralized. Workstations that are most commonly used nowadays, like SUN or VAX, are not supported.

Description

DEMOS/MP is a distributed version of the DEMOS operating system that has been operational on various architectures including a Cray 1 and is currently running on Z8000 processors connected by a local area network. A simulation was running on a VAX under UNIX.

All communication between processes is done by messages which are sent over one-way message channels called *links*. There are two types of messages, guaranteed and unguaranteed. Guaranteed messages have an at-most-once semantics achieved by an end-to-end acknowledged window protocol. The transfer of large amounts of data is facilitated by a special protocol that uses so called *link data areas* which can be accessed directly.

All links to a process are maintained by its kernel. A link can be considered as a global address that contains the identifier of the creating machine, a local unique id and the *last known machine id*. Since processes can migrate, the last field is used to localize the current host of a process resulting sometimes in a worse performance. Links are published by a name service, called *switchboard*. Each process can register at its local switchboard and make requests there which are forwarded, if necessary, to the central global switchboard. This leads to a two-level naming hierarchy.

Links are protected objects and provide much of the same function as capabilities.

A recovery facility, called *publishing* is installed which is able to recover a process in a completely transparent manner. A centralized *recorder* stores all messages that are transmitted, as well as checkpoint and recovery information. After detection of a failure, the recorder may restart all affected processes from checkpoint. All messages that were sent after the time of the checkpoint are resent to the process while ignoring duplicate messages.

Miscellaneous

Other issues: DEMOS/MP provides virtual memory with paging across the network. In this way diskless workstations can be used. Special links provide control of processes across machine boundaries.

Status: Dormant — not in use, not under development.

Contact: Barton P. Miller, Computer Science Department, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, Wisconsin 53706

References: [142], [143], [144], [145]

2.17 DOMAIN

Main Goal

The DOMAIN system is a commercial product of Apollo Computers, Inc., that connects personal workstations and server computers. All processors transparently share a common network-wide virtual memory system that allows groups of users to share programs, files and peripherals.

Advantages

DOMAIN provides an efficient network-wide single level store mechanism. However, in some cases an IPC mechanism would be preferable. Not much work is done to make the system more reliable.

Description

DOMAIN is a distributed operating system for Apollo Computers. The network is a 12 megabit-per-second token ring. Connections to other networks can be integrated. The DOMAIN user environment offers AUX, a UNIX System III compatible software-environment with 4.2 BSD enhancements. DOMAIN and AUX programs can be mixed freely.

The only way to communicate is via shared memory supported by the single level store mechanism. The object-based virtual memory allows pages to be transferred across the network. To allow recovery from link failures, the network can be installed in a star-shaped configuration.

Each object has a unique 64-bit identifier (UID) which is generated by concatenating the unique node-id of the node generating the object with a timestamp from the node's timer. The objects are located by a searching algorithm which uses hints for the most likely location. The permanent storage of an object is always entirely on one node, most often the original one. All operations on an object are performed on this node. Objects have a type UID that identifies a *type manager* handling this object. Users are provided with hierarchical names that are translated by local name servers into the object's UID. The network-wide *root* directory is replicated on every node.

Users are authenticated by the use of a replicated system-wide user registry. A user identifier consists of a person, project and organization identifier and the login node. Each file system object is associated with an *access control list* object containing a UNIX-like list of rights.

The system detects concurrency violations by the use of version numbers, but doesn't provide transparent serialization. The distributed lock manager offers the possibility of either *many readers or single writer* locks or a *co-writers* lock the latter of which allows multiple writers that have to be co-located to work at a single network node. In addition this manager is responsible for the management of the caches.

Miscellaneous

Other issues: A generic service tool provides among other things a calendar utility, a mail program and a document editor.

Diskless nodes can be incorporated.

Status: The Apollo DOMAIN system has been in full commercial production since the middle of 1981. Thousands of networks and over 20,000 workstations have been installed. The largest single network consists of more than 1800 nodes.

Contact: Paul H. Levine, Apollo Computer, Inc., Chelmsford, Massachusetts 01824

References: [146], [147], [148], [149], [150], [151], [152], [153], [154], [155], [156]

2.18 DUNIX

Main Goal

DUNIX integrates several computers connected by a local area network into a single UNIX machine providing access and location transparency. In addition, the complexity of the kernel is supposedly reduced.

Advantages

DUNIX meets its goal to provide a distributed UNIX with acceptable performance and increased availability, but nearly nothing is done to improve reliability and consistency.

Description

The system is implemented on VAX-machines connected by an Ethernet and consists of a complete new kernel. The underlying hardware has to be homogeneous. Almost all 4.1 BSD binaries run without recompilation.

The communication system uses its own protocols which provides only may-be semantics. For communication with other systems it implements the standard TCP/IP protocols

DUNIX has a single global name space for processes and files as in UNIX. Additionally the file system is partioned into groups with a *super root*, which is named '...', so each group has its own root directory ('/') providing the usual naming mechanism but the system can contain several copies of its vital files, e. g. /bin. There's not necessarily any

correspondence between physical computers and sub-trees. Each '/dev' directory names all of the system devices. Each symbolic name is mapped to a *universal file name* that has a fixed size (64 bits) and contains the host id, an index into the primary memory file table and a unique identifier.
The protection scheme is the same as in UNIX.
The availability is improved, but no measures are taken to improve reliability.

Miscellaneous

Other issues: Disks can be switched manually from a crashed computer to a running one with all files afterwards accessed in the same way.
Status: An installation is running at Bell Communications Research; some of its ideas were explored in the MOS system.
Contact: Ami Litman, Bell Communications Research, 435 South street, Morristown, N.J. 07960.
References: [157]

2.19 Eden

Main Goal

The main goal of the Eden project was to combine the distribution and integration of a collection of nodes in a network by use of an object-based style of programming. Therefore it was an important aspect to develop a special language, the Eden Programming language (EPL). It supports remote procedure calls, the use of objects and leight-weighted processes for synchronous communication. Eden is access-, location-, replication- and failure-transparent.

Advantages

The system is easy implemented and portable because it resides on top of an existing operating system. On the other hand, this results in a very poor performance. The development of distributed applications is well supported by EPL whereby proofs of correctness are facilitated.

Description

The final version is operational on a collection of SUN and VAX-machines on top of UNIX 4.2 BSD. Therefore it is possible to make use of Eden and UNIX services simultaneously. To get it to run, the UNIX kernel has to be changed by adding an IPC package, an Ethernet interface and a module to compute timestamps.
Objects in Eden mostly communicate synchronously whereby at-most-once semantics is guaranteed, but asynchronous communication is also possible. It is based on the Accent IPC (cf. section 2.1).
Eden objects have an active form (a UNIX process) and possibly a passive form (stored on disk). To locate an object, which can change its location even during an access, caching combined with a special search mechanism is used.
Each object is given a reference by a protected capability which contains a unique identifier and a field with specific rights.
To improve reliability, an object can checkpoint autonomously so that it can be restarted after a crash. This is done automatically if the object is invocated and no active form exists. There are two ways to replicate objects: Either the whole objects are replicated or only the passive forms. To preserve consistency, a form of Gifford's weighted voting scheme is used. Neither form of replication copes with network partitions. Additionally there is the need for a special transaction manager.

Miscellaneous

Other issues: A garbage collection removes objects without capabilities but this can only work if all objects are passive. To demonstrate the properties and behavior of an object, each has a concrete Edentype which describes all possible invocations.
Status: The project is completed and has been in use for several applications.
Contact: E.D Lazowska, Department of Computer Science FR-35, University of Washington, Seattle, WA 98195

References: [158], [159], [160], [161], [162], [163], [164], [165], [166], [167], [4], [29]

2.20 EFS

Main goal

The main motivation behind the MASSCOMP (Massachusetts Computer Corporation) EFS approach was the desire to bring full transparent access to files on remote computer to all users. The goal of transparency has been achieved by building the concept of remoteness into the existing UNIX file system I/O architecture.

Advantages

EFS is a system that operates in a multiprocessor environment. In its 1985 form it provides transparent remote access only to disk files. Remote devices and diskless nodes are not supported. The concept of process pools, i. e. available kernel processes, which can play agent roles for any EFS client request, is the main advantage of EFS. No single client process is able to block an EFS server from serving other client processes.

Description

EFS is client/server based and operates between two or more MASSCOMP computers on the same Ethernet, all running an enhanced version of the Real-Time UNIX (RTU) kernel, a MASSCOMP variant of the UNIX time sharing system. A new *reliable datagram protocol* (RDP) has been designed and implemented providing guaranteed message delivery and the ability to dynamically create multiple low-overhead connections through a single socket. The IPC uses the 4.2 BSD socket mechanism.

Because EFS works in an UNIX environment, an inode contains all information with regard to a local file. A *rinode* is a special variant of an inode. It contains a machine-ID which uniquely identifies a single host on the network as well as the address of an usual in-core inode within the memory of that host. Whenever a remote file is involved, the rinode information is used. Mounting enables network-wide access to remote files.

Protection across the EFS is built in as follows: All the machines in the network are converted to the same password file, so that nearly the normal UNIX protection mechanisms can be used with the exception of selective mount points. These are added to make certain files on some machines inaccessible to non-authorized users. The global protection domain is used to simplify the concept of file ownership.

The client/server model in EFS is stateful. All remote file operations are based on transactions. The client process is blocked until the server completes the transaction.

Miscellaneous

Status: Research is going on to eventually support remote devices and diskless nodes.

Contact: C. T. Cole, Ardent Computer Corporation.

References: [168]

2.21 Emerald

Main Goal

In the first place, Emerald is an object-oriented language. It is also an approach to simplify distributed programming, though, as well as to build a distributed operating system through language support while providing high performance and flexibility, both locally and in the distributed environment. Emerald provides access and location transparency.

Advantages

An advantage of Emerald objects is its capability to create small (e. g. integers) and large (e. g. compiler) objects in the same way using the same object definition mechanism. A single semantic model suffices to define them. A disadvantage is that performance reasons force a weaker understanding of transparency. Some aspects of distribution are not generally hidden from the programmer. A language support exists for the explicit notions of location and mobility.

Description

A prototype implementation for a compiler of the Emerald language and a small run-time system have been constructed on top of the DEC Ultrix operating system. Emerald itself runs on a 10Mbit/second Ethernet local area network of DEC Microvax II workstations.

Objects are the units of programming and distribution, and the entities between which the communication takes place. Each object exports a set of operations and can be manipulated only by the use of these operations. All information is encapsulated in specific objects.

Objects communicate through invocation. Since parameters are objects to themselves, the method of parameter passing is *call-by-object-reference* (similar to CLU, Argus, cf. section 2.5). In Emerald, objects are mobile (process migration in non-object systems). This mobility enables Emerald to move parameters to the callee (*call-by-move*) so as to avoid remote references (Argus needs parameter passing by value).

Emerald is typed. Objects are typed abstractly, which means that an interface to a set of operations is supported as well as their signature and their semantics. The signature includes all of the information about the operation, such as operation name, and the number, names and abstract types of arguments and results.

Emerald supports inherent parallelism and concurrency, both among objects and within an object. Processes have exclusive access to objects through monitored variables and may synchronize using traditional condition variables.

Availability in Emerald. When an object moves, it leaves behind it a so called forwarding address so that invocations sent to the old location can be forwarded correctly. In the case a machine crashes, forwarding addresses are lost. Emerald solves this problem by an exhaustive search for this object, implemented as a *reliable broadcast protocol.*

Security issues, replication and failure handling are not mentioned due to the current status of the project.

Miscellaneous

Other issues: For reasons of performance a programmer is enabled to determine the location of objects. An application can collocate objects that communicate intensively to reduce communication overhead. Alternatively, concurrent subcomputations can be placed on different nodes to balance the load.

Status: A prototype implementation is constructed. Research is going on to implement the features of object mobility and distributed garbage collection.

Contact: Andrew Black, Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195.

References: [169], [170]

2.22 GAFFES

Main Goal

GAFFES (Global, Active and Flexible File Environment Study) is a study of a globally distributed file system designed to share information in a world-wide network consisting of more than a million workstations. GAFFES, designed at the University of California in Berkeley, will provide location, access, concurrency and replication transparency to a huge user community.

Advantages

GAFFES is highly distributed in order to promote high capacity and modular growth. It provides independent services such as naming, replication, caching, security, and authentication. GAFFES is a serious approach to build a globally-distributed file system in a huge world-wide communication network.

Incidentally, GAFFES is an experimental design. It neither has been built, nor is it intended to be built.

Description

GAFFES could run on future networks varying in speed and architecture, such as low-speed telephone lines and high-speed optical fibers. The client workstations connected by this network would be quite heterogeneous, ranging in power from PCs to large mainframes and running many different operating systems. The heterogeneity extends

into administration. The network crosses national boundaries and is managed by different organizations resulting in difficult security and privacy issues.
GAFFES was designed to be built on a base of traditional file servers and RPC communication. RPC communication can be performed between any two machines in the network.
GAFFES files have unique names which are independent of a user's location or the file's location or the number of replicas. Moreover, clients may identify files with descriptive names allowing file accesses based on their contents. The GAFFES name space is hierarchical. Each component of a file name further qualifies the object in the name space.
The file system is designed to provide a version history of files. Any kind of manipulation is supported on the version tree. New versions are created with each modification, and the naming mechanism insures that clients always access the latest version as far as *unqualified* requests are made. Unqualified requests do not contain a version number for any particular file.
Both client and server machine have a globally-unique identifier. Both contain hardware encryption devices which can be used in security procedures. On demand, public-key/private-key pairs can be used for secure data transmission. Each organization has autonomous control over the security of its own files. The rights to user access are freely granted or revoked. A GAFFES security system provides a uniform mechanism for authenticating users and ensures the privacy of data.
The creator of a file is responsible for the degree of replication. Thereby he determines the availability of the new file. Each client performing a operation specifies whether caching should be used and what degree of consistency is required for the cached copies.
All client requests are performed as part of a transaction which is run as a set of atomic serializable actions. Nested transactions are supported by a traditional file server interface.

Miscellaneous

Status: GAFFES is just a design, not an implementation. GAFFES is a first step towards the design of a large-scale globally-distributed system.
Contact: Douglas Terry, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Ca. 94720. Now at Xerox PARC.
References: [171], [172]

2.23 Grapevine

Main Goal

Grapevine is developed at Xerox Palo Alto Research Center as a distributed, replicated system that provides message delivery, naming, authentication, resource location and access control services in an internet of computers. Access to the servers as well as location and replication of entries are transparent to the users.

Advantages

Grapevine is a reliable and highly available mailing system by replication of function among several servers. However, this leads in some cases to unexpected delays and errors. Several details in implementation will have to be changed in order to efficiently serve the proclaimed 10.000 users.

Description

The Grapevine servers are running on Alto computers and 8000 NS processors with the Pilot operating system. Client programs run on various workstations under several operating systems.
The communication is built on internet protocols. Either individual packets can be sent (unreliable) or byte streams may be established with the PUP protocol, which are sent reliably and acknowledged.
Each entry in the registration database has a two-level hierarchical name, the registry being the first part (corresponding to locations, organizations or applications), thus dividing the administrative responsibility, and the

name within that registry as the second part. The choice of the servers is done by programs on the client side, by choosing the nearest available server.
Authentication at associated file servers is done by the use of Grapevine. In this manner the protection scheme contains individuals and groups. To improve performance, the file servers cache access control checks.
Replicated submission paths are available, because each server can deliver messages to any user and each user has inboxes on at least two servers. Registries are replicated at various registration servers as a whole. Each registry is administered independently of the others. Clients have to cope with transient inconsistencies. Any conflicting updates of registration entries do not get solved.

Miscellaneous

Other issues: Grapevine is also used to control an integrated circuit manufacturing facilities.
Status: Grapevine is in use by Xerox, where the system should consist of up to 30 servers and 10.000 users (Summer 1983: 17 servers and 4400 users). The registration service became a product of Clearinghouse service (cf. section 4).
Contact: Michael D. Schroeder, DEC Systems Research Center, 130 Lytton Ave., Palo Alto CA 94301
References: [173], [174]

2.24 Guide

Main Goal

Guide (Grenoble Universities Integrated Distributed Environment) is an object-oriented distributed operating system for local area networks. It embodies the object-oriented architecture defined in the COMANDOS Esprit Project (Construction and Management of Distributed Office Systems). The main design decisions comprise first of all the integration of an object model to a language which includes the notion of type and classification with simple inheritance and second, the dynamic migration of multi-threaded jobs among network nodes. Guide provides location transparency of network resources to the application layer.

Advantages

Guide objects are usually small (like files in traditional systems) and may be combined to form larger units. This granularity allows locking of smaller units and a higher concurrent throughput (compared to guardians in Argus or Ejects, Eden Objects in Eden which are usually larger units, such as servers). Implementing Guide eventually as a new kernel will increase performance. Since the implementation of the system is still in progress, more criticism is not valid.

Description

The Guide operating system is based on a set of heterogeneous workstations, such as Bull SPS-7 and SPS-9 and SUN-3, connected by a high-speed Ethernet-based local area network. Guide is implemented on top of the System V version of UNIX.
Guide uses persistent objects, i.e. the object's lifetime is not related to the execution of programs, but the object is in existence as long as at least one other persistent object refers to it.
Each object is named by a system-wide unique identifier. Objects encapsulate a set of data and a set of procedures. A *type* describes the behavior shared by all objects of that type. A *class* defines a specific implementation of a type. Classes are used to generate instances. To define a specialization of a given type, subtyping can be done. The hierarchy of subtypes among types is paralleled by the subclassification of classes. A subclass inherits the operations of its superclass, and can overload them. Generic types and classes are also provided.
Objects are only accessible through typed variables which are used to invoke operations on objects.
A similar concept to the MACH's *task and thread* approach is the *job and activity* concept. A *job* is a multiprocessor virtual machine which provides an arbitrary number of sequential threads of control, called *activities*. Activities communicate through shared objects.
A nested transaction concept is implemented which provides exactly-once semantics to atomic objects. The property of an object to be atomic is fixed at the time the object is created. Each transaction is entirely contained within

a single activity. The classical two-phase locking scheme is applied to atomic objects which are the locking units. Security and availability issues are not discussed due to the current status of the project.

Miscellaneous

Other issues: The Guide language.

Status: Guide is a joint project between the Laboratoire de Génie Informatique and the Bull Research Center at Grenoble. Both organizations work in *COMANDOS*.

The main design is completed. A prototype single-node implementation is being done on top of the UNIX System V.

Contact: S. Krakowiak, Laboratoire de Génie Informatique, IMAG Campus, BP 53 X, 38041 Grenoble Cédex, France.

References: [175], [176], [177], [178], [179]

2.25 Gutenberg

Main Goal

Gutenberg is an operating system kernel designed to facilitate the design and structuring of distributed systems and to provide all forms of transparency. The crux of the Gutenberg approach is its use of port-based communication, non-uniform object-orientation and decentralized access authorization using ports and a *capability directory*.

Advantages

Problems of performance within systems that support protection should be solved in Gutenberg by structuring objects at the operating system level only if they are shared. Objects are assumed to be large to amortize the cost of communication. Experimentation with the kernel will eventually determine the feasibility of this approach.

Description

The Gutenberg kernel was supposed to be built on top of UNIX, but nothing has been published about necessary changes to the UNIX kernel or its underlying hardware.

Processes need ports to communicate with each other. There are three types of ports (send, receive, send/receive). Messages are sent synchronously (with acknowledgement), asynchronously or RPC-like.

The creation and use of ports are controlled by the use of capabilities. All capabilities that grant access to potentially sharable objects are listed in the capability directory and are further organized into groups, called *cd-nodes*. Cd-nodes are identified by system-wide unique names for the kernel's use and by user-specified names. Cd-nodes are linked to other cd-nodes via capabilities. Capabilities in cd-nodes are stable and sharable. Each process has an associated cd-node and a set of transient capabilities, both of which are owned by this process and are destroyed when the process terminates.

Each creation of and access to user-defined objects involves the checking of kernel-defined capabilities. Therefore, authorization is guaranteed if each kernel is trusted. Capabilities can be transferred permanently or temporarily. Each transfer is monitored by the kernel.

The capability directory is partitioned and replicated on each site. Cd-nodes are replicated in locations at which they might be used. The site a cd-node is created at keeps track of the sites with copies of the same cd-node.

Synchronization of access as well as consistency of the replicas are preserved in a *two-phase-locking scheme*, either with a primary site or by a distinction between read and write. Gutenberg provides a means to decompose the execution of processes into units of computations (called *recoverable actions*) that execute atomically with respect to kernel objects and stable storage. Communication leads to dynamically developed dependencies between these actions. Stable storage and a two-phase-commit protocol are used to ensure atomicity.

Miscellaneous

Status: The kernel is currently being implemented.

Contact: David W. Stemple, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003

References: [180], [181], [182], [183], [184], [185], [186], [187], [188]

2.26 HARKYS

Main Goal

The main goal of the HARKYS project was to provide a network file system both location transparent and reliable for different machines running different UNIX systems without modification to the kernel.

Advantages

HARKYS is easily portable and makes the joining of UNIX systems as different as 4.2 BSD and System V in one global file system possible. But, locking problems will have to be solved subsequently.

Description

HARKYS can only be installed above an existing UNIX system. Programs using remote files must be relinked with a new C-library, but without any change to the sources. Compatibility with 4.x BSD, the System V family, AIX and ULTRIX is given. HARKYS is presently running on various machines, e. g. VAX and SUN.

RPCs are used for client/server communication based on standard protocols. Up to now TCP/IP is supported. A software layer is placed between the user and the operating system interface which forwards remote requests to the corresponding server processes. These are created by a daemon, called *spawner*, for each participating operating system separately. In addition, the server process makes any data conversions required.

Remote hosts need to be mounted. If a path name contains a mount point, the translation of a new path name starts from the remote root directory. This way, the remote mount point has to be the root.

For the purpose of access control, the standard UNIX mechanism is used although more general access strategies can also be employed. Rights to other hosts are described in an authentication database which is included at each host.

The client process maintains all information concerning remote files, some information being replicated at the server's side. In the case of a crash of a process or a host or in the case of network problems, all remote and local resources are released with the help of this data, and participating processes are terminated.

Miscellaneous

Status: The system has been tested on VAX/750 and SUN systems. Future work includes the extension of accounting mechanisms across the network, the utilization of other transport protocols, and the provision of system administrator network tools.

Contact: A. Baldi, Research & Development Division, Systems & Management S.p.A., Pisa, Italy.

References: [189]

2.27 HCS

Main Goal

The Heterogeneous Computer Systems project (HCS) was designed to integrate loosely-connected collections of hardware and software systems. The costs of adding new types of systems should be decreased, and the number of common high-level services that users can hope to share should be increased. These include mail, filing and remote execution. HCS provides access and location transparency.

Advantages

The main advantage is its capability to incorporate a new system at low cost and without masking the unique properties for which it was obtained. Only prototypes of the proposed services exist, so that not much can be criticized yet nor said about its applicability.

Description

Prototypes of the HCS services are being tested on various hardware and software systems including SUN, VAX, IBM RT-PCs and XEROX machines, UNIX, VMS and XEROX operating systems and the BIND and Clearinghouse

name services. The unique communication mechanism is the heterogeneous remote procedure call (HRPC) which emulates the native RPCs of the underlying systems by specifying interfaces to the five principal components of an RPC: the stubs, the binding, transport and control protocol as well as the data representation. These components are separated from each other and can be selected dynamically in order to achieve flexibility at the cost of additional processing at bind time. The performance of the HRPC is essentially identical with that of native RPCs (tested for the SUN and Courier RPC).
The name service (HNS) provides a global name space with uniform access by the use of the local name services. Each name contains a context which specifies the name service and an individual name which determines the local name on that service. The name-service-specific code is encapsulated in a set of HRPC-accessible routines, which can be added easily without any recompilation of existing codes.
Neither security issues nor measurements for reliability and performance improvements play any role.

Miscellaneous

Other issues: The prototype of a filing service (HFS) has been built. It is composed of a file server, a type server managing type information of the files and a mechanism for generating routines for the access to the HFS. A mail service has been developed which coexists with existing mail services by a *global alias* service that contains entries for each user. The HCS environment for remote execution (THERE) is a facility for constructing remote execution clients and servers in a net. Both client and server make use of service specific interpreters.
Status: Prototypes of the services are being tested at the University of Washington. Research is still going on.
Contact: David Notkin, Department of Computer Science, FR-35, University of Washington, Seattle, WA 98195
References: [190], [191], [192], [193], [194], [195], [196]

2.28 Helix

Main Goal

Helix is an object-oriented distributed file system for the XMS system developed at BNR in Ottawa. Its main objectives are the support of a diversity of applications, capability-based security, and the provision of access, location and failure transparency.

Advantages

Helix has some useful features for failure resistance and protection of files. The server cpu could become a bottleneck since the server has to do most of the work (e. g. path name translation, check of access rights). The system seems to be configured solely for the software development at BNR, since it is hardly portable to other systems.

Description

The XMS system is being run on Motorola MC68010-based workstations. Helix is compatible with the UCSD file system.
Helix uses the XMS blocking communication primitive, called *rendezvous*, but nothing has been published about the underlying protocol and the semantics of a rendezvous. The communication between multiple LANs is done by special communication servers and is based on the ISO OSI model. Helix is extended to multiple LANs by a server called network Helix which implements the standard interface and passes requests to the far end via virtual circuits.
Each object has a unique capability containing the server-id, a field of rights, the number of the object and some random bits to make it unforgettable. Capabilities are stored in directories together with a user-defined string. Directories can be linked freely, not necessarily hierarchical. The naming of objects can be done either directly by a capability or by a path name. Requests for remote files are passed from a so called *client Helix* to the remote server that does path name translations and capability checks. Files are grouped in logical volumes, typical equivalent to file servers. Links between different volumes are not permitted. Caching is used to improve the performance.
The capabilities which are created by the system are automatically encrypted and decrypted.
The system allows transactions, which can be used for updates, and the creation of instances for a file. Transactions

can also be applied on sets of files or otherwise can be nested. Some measures are taken to improve reliability: Unreadable loops which are possible in a non hierarchical name space are discovered and linked into a special directory for manual resolution. In case of program failures, all files are closed or aborted. Clients are periodically checked to detect crashes.

Miscellaneous

Status: The system is in use at BNR with 15 LANs and close to 50 file servers which are supporting almost 1000 workstations.

Contact: Marek Fridrich, Bell-Northern Research, PO Box 7277, Mountain View, CA 94039

References: [197], [198], [199]

2.29 IBIS

Main Goal

The purpose of IBIS is to provide a uniform, UNIX compatible file system that spans all nodes in a network. IBIS is being built as part of the TILDE project (Transparent Integrated Local and Distributed Environment, cf. section 4) whose goal it is to integrate a cluster of machines into a single, large computing engine by hiding the network. IBIS, a successor of STORK, provides four levels of transparency: file access, location, replication and concurrency.

Advantages

IBIS exploits location transparency by replicating files and their migration to the place where they are needed. Replication and migration of files improve file system efficiency and fault tolerance. The main disadvantage of IBIS is quite an anomaly. Since IBIS is implemented as a separate layer between UNIX kernel and the applications, for local file accesses, the overhead rises with the size of the file resulting in a poor performance. Placing IBIS into the kernel should speed it up.

Description

IBIS runs on VAX-11/780 hosts under the operating system UNIX 4.2 BSD connected via a 10Mbit/sec Pronet. Because IBIS implements all remote file operations with the IPC facility of UNIX 4.2 BSD and is based on the standard TCP/IP protocol, it is not restricted to LANs. Even if a remote host running TCP is connected via gateways or lossy subnets it can participate in IBIS. The access transparency layer is sandwiched between UNIX kernel and its applications. For remote files, this layer uses a RPC protocol for interacting with the server. Not the whole UNIX system call semantics is implemented by IBIS. Some UNIX commands are missing, some are replaced by look-like UNIX commands. For example, *rm* does not remove a file. Instead, it moves the file to a special directory called *tomb*, from where it will eventually be removed. With symbolic links crossing machines, a user can construct a directory tree spanning several machines.

IBIS provides UNIX like file names (path name strings) in a hierarchical name space.

Security issues in IBIS are implemented via authentication. Both sites, client as well as server, need a way of ascertaining the authenticity of each other. IBIS solves this problem as follows: A dedicated server is established by an authentication scheme called a *two-way handshake via secure channel*. A client reserves a port number and executes a so called *connection starter* program. This starter opens a secure channel to the so called *server creator* in the remote host and passes the user ID to it. The server creator process establishes a server with access rights, as indicated by the user ID and reserves a port number as well. Exchanging port numbers via the secure channel from now on allows the establishment of a connection verifying each other's portnumber. The secure channel is implemented by making both, connection starter and server creator privileged processes which communicate via privileged port numbers.

Availability in IBIS is achieved by replication. There are two main differences with regard to LOCUS (cf. section 2.31). IBIS avoids both the bottleneck of a synchronization site (LOCUS's CSS) as well as remote synchronization for local replicates. To achieve this, IBIS uses a decentralized control scheme. A primary copy strategy is used with

the following update protocol. If the primary copy is updated, the update broadcasts a signal that invalidates all cached copies. The use of strict state transition rules provides consistency. For further information please confer the references.

Miscellaneous

Other issues: Migration is used to relocate files so as to speed up write requests (demand/forced replication). IBIS provides explicit commands for migration.

Status: Research is going on to develop automatic migration and placement of files.

Contact: D. Comer, Department of Computer Science, Purdue University, West-Lafayette, IN 47907.

References: [200], [201], [202]

2.30 JASMIN

Main Goal

JASMIN is an experimental distributed operating system kernel that provides message passing over dynamic communication capabilities (location transparency). It is in use as a distributed processing research tool at Bell Communication Research.

Advantages

JASMIN is a merely academic approach meant to demonstrate and evaluate some interesting fields in distributed environments. Therefore, too much criticism out of place.

Description

JASMIN functions as a client/server model. It consists of a new operating system kernel with an execution environment of servers and a connection to an UNIX host computer. The JASMIN kernel provides IPC, tasking and scheduling, whereas other services are provided by server tasks separate from the kernel.

Communication: In JASMIN, communication between so called *tasks* is via messages sent on *paths*. Messages are sent asynchronously. The sender does not block as in the typical client/server model. The kernel guarantees that messages are reliably delivered in the order they are sent in.

Security: Paths are one-way communication links that grant the holder a capability to send messages, similar to the links of DEMOS/MP (cf. section 2.16) or Roscoe (cf. section 4).

Naming: JASMIN actually maintains a two-tiered hierarchy of name servers: a single global name server (GNS) and multiple local name servers (LNS). LNSes have a path to the GNS. LNSes determine whether a server is local or remote.

Miscellaneous

Other issues: A distributed database management system is a major application of JASMIN.

Status: Research is going on to build reliable disk servers on top of two normal JASMIN disk servers.

Contact: H. Lee, Bell Comm. Research, Morristown, N.J.

References: [203], [204], [205]

2.31 LOCUS

Main Goal

LOCUS was developed to provide a distributed and highly reliable version of an operating system to the huge UNIX-community. LOCUS tools support an automatic conversion from stand-alone UNIX nodes to network-wide connected LOCUS nodes simultaneously offering a high degree of transparency, namely location-, concurrency-, replication- and failure transparency.

Advantages

LOCUS' main advantage is its reliability and availability feature supporting automatic replication of stored data, with the degree of replication dynamically under user control. Due to a majority consensus approach within par-

titioned subnets to build up a temporary still-connected partition, the system remains operational with a high probability. This feature implies that sometimes out-of-date data will be modified resulting in an inconsistent global state. The tradeoff between performance, remaining operational as well as consistency leads to LOCUS' disadvantage: Even when a merge-protocol frees a user in most cases from consistency checks after network partitionings, sometimes user interaction can still be required in order to get back into an up-to-date global consistent state.

Description

LOCUS is a step-by-step modification and extension of the UNIX operating system compatible with the BSD and System V branches. Load modules of UNIX systems are executable without recompiling. Tools exist for an automatic conversion to LOCUS by exchanging the UNIX kernel with a so called LOCUS kernel and by reformatting an existing UNIX file system without loss of information. The 1983 version of LOCUS supports DEC VAX/750, DEC PDP-11/45 and IBM PCs connected by Ethernet or Token-Ring LANs. The communication between different LOCUS kernels uses a point-to-point connection (VC, virtual circuit).

A single tree structure spanning all nodes of the network includes virtually all the objects of the file system. All objects possess a global unique name in a single uniform and hierarchical name space, identified by a path name (character string). There is only one logical root in the entire network. The path names do not include information about the location of objects. So called *file groups* are self-contained subtrees within the naming hierarchy.

Security in terms of intruder defense or secure data transmission plays a minor role and the mechanisms are comparable to the ones implemented in UNIX.

File replication is made possible in LOCUS by multiple physical containers for each logical file group, and gaining access to a replica is done according to a certain protocol: A centralized synchronization site, the so called CSS (*current synchronization site*), decides whether a request from a US (*using site*) is possible or not (is a copy available?), and determines the actual SS (*storage site*) needed to satisfy this request. An atomic commit using shadow copies of updated pages provides consistency in cases of crashes or network failures. Nested transactions can be made. Token passing for distributed operations on the file system permits serialization.

Miscellaneous

Other issues: remote tasking is supported by two new (remote process) system calls, *migrate* and *run*.

Status: Originally a research effort, at UCLA in the late 1970s, LOCUS nowadays has been transferred to a commercial venture: LOCUS Computing Corporation, Santa Monica, Ca.

Contact: Gerald Popek, Department of Computer Science, UCLA, Los Angeles, Ca.

References: [206], [207], [208], [209], [210], [211], [212], [213], [214], [77], [215], [216], [217]

2.32 Mach

Main Goal

Mach was designed with the intention of integrating both distributed and multiprocessor functionality. In addition full binary compatibility with UNIX 4.3 BSD should be given by preserving a simple, extensible kernel that runs most of the services, e.g. file service, in user-state code. It is an extension of the Accent system to multiprocessors and other types of hardware, providing the same classes of transparency and adding some further features.

Advantages

Mach is a good environment for distributed and parallel applications because of its variety of inter-process communication possibilities, ranging from shared memory to efficient message passing for uni- and multiprocessors. Moreover it provides a good chance for the huge UNIX community to change to a real distributed system without giving up their convenient services. Although Mach is more extensive, its performance is in line with the Berkeley UNIX. On the other hand not much work is done on built-in failure handling and reliability improvements.

Description

Machs current implementation runs on all VAX uni- and multiprocessors, SUN 3s, IBM RT-PCs, the Encore Multimax with up to twenty NS 32032 series processors and the Sequent Balance 21000. As mentioned above, it is fully binary compatible with UNIX 4.3 BSD.

Mach uses a simplified version of the Accent IPC facility because leight-weighted processes (threads) exist that are able to handle some forms of asynchrony and failures more efficiently. Unlike BSD, there is one interface, the port, which is consistent to all resources. See section 2.1 for more information concerning communication, naming and protection.

In addition, MatchMaker, an interface specification language, enables interfaces between clients and servers to be specified and generates remote procedure call stubs. These interfaces are able to perform run-time type-checking and deliver enough information to the network servers to make data-type conversions that exchange between different types of hardware.

Many of the missing reliability issues are included in a system called Camelot (cf. section 4), which relies on Mach and is still in development.

Miscellaneous

Other issues: It is possible to share memory between multiple threads (the entities of control) and tasks (the entities of resource allocation) in a highly machine-independent manner, providing good performance by the use of copy-on-write virtual memory. An adb-like kernel debugger is available.

Status: Mach is running at Carnegie Mellon University, supporting the Andrew distributed file system. It is free for distribution and an export license should be available by September 1988. Development and improvement are still in proces and new versions will be available to all users.

Contact: Richard F. Rashid, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213.

References: [218], [219], [220], [221], [222], [223], [224], [225], [6], [226], [227], [228], [229], [230], [231], [232]

2.33 Medusa

Main Goal

Medusa is a distributed operating system designed for the Cm* multimicroprocessor. It is an attempt to produce a system that is modular, robust, location transparent, and to take advantage of the parallelism presented in Cm*.

Advantages

Each Cm contains an LSI-11 processor and 128 kB of main memory. This hardware constraint leads to the following design decision. Medusa discards the assumption that every processor in the system must be able to fulfill all services within the operating system. The entire operating system is therefore divided into disjoint utilities resulting in sometimes "unnecessary" non-local executions. On the other hand, utilities are a good example of how an operating system can be distributed as a whole.

Description

Medusa is the successor to the StarOS operating system (cf. section 4). StarOS was the first operating system developed at the Carnegie-Mellon University for the Cm* multimicroprocessor computer.

Medusa's functionality is divided into disjoint *utilities* due to the small main memory of the Cm internal LSI-11 processor. Just a small interrupt handling kernel is distributed among all processors.

There is no guarantee given that any particular processor contains a copy of any particular utility. Besides, a processor is only allowed to execute a local residing utility. In Medusa, messages therefore provide a *pipe*-based mechanism for synchronous (point-to-point) cross-processor function invocations. Message pipes contain no information as to location.

Each utility protects its own execution environment. It may migrate around the system, once the format for message exchange via pipes has been selected.

Medusa provides an object-oriented model. Each object is manipulated by type-specific operations. Access to objects is protected by descriptor-lists. An important object class is the semaphore class which provides concurrency control for shared objects. Deadlock detection is supported.
Concurrency support is achieved by the so called *task force* concept. This pertains to a collection of concurrent *activities*, entities that actually get scheduled for execution (analogous to processes in traditional systems). All activities of a given task force can simultaneously run (almost all the time).
Failure handling, replication and availability issues are not addressed.

Miscellaneous

Status: The design for Medusa was begun in the late fall of 1977, and coding started in the summer of 1978. The current status of the research is unknown.
Contact: John K. Ousterhout, Dept. of Computer Science, UC Berkeley
or Donald A. Scelza, PRIME Computer, Inc., Old Connecticut Path, Framingham, MA 01701.
References: [233], [234], [235], [236]

2.34 Meglos

Main Goal

Meglos extends the UNIX operating system with simple and powerful communication and synchronization primitives needed in real-time environments. Location and access transparent communication is achieved using channels.

Advantages

The main advantage of the Meglos operating system is that it can be used in real-time environments. The major constraint for a Meglos user is the hardware as described below.

Description

The Meglos operating system is implemented as a new, UNIX extending kernel. Meglos consists of up to 12 processors using M68000 based Multibus computer systems. Each processor is connected to the 80 MBit/second S/Net. A DEC VAX computer running UNIX serves as a host processor for the system. The satellite processors are built around the Pacific Microsystems PM-68k single-board computer.
Meglos is an exact emulation of the UNIX environment allowing programs to make use of I/O redirections and pipes. Programs running under Meglos are controlled from the UNIX system which provides access to files.
A specialized IPC has been designed to support real-time applications. The IPC mechanism is based on communication paths called *channels*. Each channel permits bilateral communication via message exchange. Independent flow-control is provided for every channel to meet real-time constraints.
Channels are named and provide location as well as access transparent communication, as well as synchronization between readers and writers.
For real-time response, a priority-based preemptive scheduler and an efficient process switching has been implemented.
Security issues, failure handling and availability of the system are not mentioned.

Miscellaneous

Other issues: MIMIC, a robot teaching system has been implemented under Meglos using real-time features. Also, Meglos includes a symbolic debugger for C programs.
Status: Several prototypes of Meglos systems have been built and are being used to support multiprocessor robotics applications in the AT & T Robotics Research Department.
Contact: Robert D. Gaglianello, AT & T Laboratories, Holmdel, NJ 07733.
References: [237], [238], [239], [240], [241]

2.35 MOS

Main Goal

The Multicomputer Operating System (MOS) is a distributed UNIX system. It is a general-purpose time-sharing operating system which makes a cluster of loosely connected independent homogeneous computers behave as a single-machine UNIX system. The main goals of MOS include location and access transparency, decentralized control, site autonomy and dynamic process migration.

Advantages

MOS is one of the few systems that attempts to integrate load-balancing philosophies into a distributed operating system. Process migration and a suboptimal load-balancing algorithm are implemented.

Description

MOS was developed at the Hebrew University in Jerusalem and consists now of seven PCS/CADMUS 9000 machines (with MC68010 CPU) interconnected through a 10 Mbit/sec Pronet local area ring network. The machines must be homogeneous in order to allow process migration. Diskless nodes are supported.

MOS is an enhanced UNIX V7 kernel supporting all UNIX system calls. Thus, MOS is binary compatible with UNIX. MOS uses the RPC mechanism based on the Unreliable Datagram Protocol for communication.

The file system in MOS is a forest. Each tree is a complete UNIX file system. A single file system is achieved by placing a *superroot* above all trees of the forest. Standard, path named, hierarchical UNIX naming conventions are provided.

Security issues extending UNIX features are not implemented. An outside-kernel software exists that provides replication of storage to increase the system's availability.

No provisions are made for failure handling. MOS does not support atomic file transactions since these features are not supported by UNIX.

Miscellaneous

Other issues: The MOS Multicomputer Operating System is a distributed implementation of UNIX providing *suboptimal load balancing* by means of process migration. Each machine stores load information from a fraction of other machines, including itself. Periodically the recent half of that information is transmitted to a machine which is chosen at random. The transmitted information replaces the "stale" half of the load statistics previously known from the receiving machine. Processes always migrate to underloaded nodes and are executed remotely. In addition light-weighted processes exist which can be executed on different machines.

Status: A PDP-11 version of MOS has been in operation since 1983, the current version is operational since Spring 1988. Research is going on in performance modeling, reliability, parallel algorithms and distributed applications.

Contact: Amnon Barak, Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel.

References: [242], [243], [244], [245], [246], [247], [248], [249], [250]

2.36 NCA/NCS

Main Goal

The Network Computing System (NCS) is a portable implementation of the Network Computing Architecture (NCA), a framework for developing distributed applications. It supports various machines running different operating systems. It is designed to represent the first step in a complete network of a computing environment by providing all aspects of transparency.

Advantages

NCA/NCS eases the integration of heterogeneous machines and operating systems into one computing environment thus expanding the set of network applications. A comparison with similar systems (e.g. HCS, cf. section 2.27) is difficult since performance hasn't been measured.

Description

NCS currently runs under Apollo's DOMAIN/IX, UNIX 4.x BSD and SUN's version of UNIX. Implementations are in progress for the IBM PC and VAX/VMS.

The system supplies a transport-independent remote procedure call facility using BSD sockets as the interface to any datagram facility, e.g. UDP. It provides at-most-once semantics over the datagram layer, with optimizations if an operation is declared to be idempotent. Interfaces to remote procedures are specified by a Network Interface Definition Language (NIDL). NCS includes a portable NIDL compiler which produces stub procedures that handle data representation issues and connect program calls to the NCS RPC runtime environment.

Objects are the units of distribution, abstraction, extension, reconfiguration and reliability. A replicated global location database helps locate an object, given its universal unique identifier (UUID), its type or one of its supported interfaces. UUIDs are an extension of DOMAIN's UID (cf. section 2.17).

A Distributed Replication Manager (DRM) supports a weakly consistent and replicated database facility. Weak consistency means that replicas may be inconsistent but, in the absence of updates, converge to a consistent state within a finite amount of time.

A simple locking facility supports concurrent programming.

Miscellaneous

Other issues: A Concurrent Programming Support (CPS) provides leight-weight tasking facilities. A data representation protocol defines a set of data types and type constructors which can be used to specify sets of typed values which have been ordered.

Status: Apollo has placed NCA in the public domain. Implementations are currently in progress for further machine types. A future version should include a general purpose name server that will be able to find an object by attributes rather than by text name. A transaction facility and strongly consistent replication is additionally planned.

Contact: Nathaniel W. Mishkin, Apollo Computer, Inc., Chelmsford, MA. 01824

References: [251], [252]

2.37 Newcastle Connection

Main Goal

The Newcastle Connection or UNIX Unite or NC for short was developed at the University of Newcastle upon Tyne, England, to connect UNIX or UNIX look-alike systems in a way to achieve location transparent access among network resources.

Advantages

NC was one of the first distributed file systems that became operational in an academic environment. With NC many students in the early days of distributed systems had the opportunity of taking a close look at an active system.

Not all commands are supported such as *su, pwd.*

Description

NC ran on a set of DEC PDP-11 computers under UNIX System V7 connected via the Cambridge Ring local area network. NC is implemented as a new layer on top of an existing UNIX operating system. Thus, no modifications were necessary to any existing source code, either of the UNIX system, nor of any user program.

Its role is to filter out system calls that have to be redirected to another UNIX system, and to handle remote system calls that are directed to it from another UNIX system. Communication between two Newcastle connection layers is based on a reliable Remote Procedure Call protocol in a client/server relationship.

The major issue of design was the superroot. A file name consists of the superroot and a trailing UNIX file system path name. If a program is involved, it is executed at whatever site the file store is held in. Data is transferred between sites in response to normal UNIX read or write requests.

The local system administrators are responsible for allocating ordinary user-IDs and for maintaining a table of recognized remote user-IDs. Unrecognized remote user-IDs can be mapped to restricted "guest"-IDs. Access to the whole NC UNIX file system is via the simple conventional login. No other security issues are implemented. Neither replication, load balancing among the nodes, nor fault tolerance issues are implemented.

Miscellaneous

Status: Research is currently aiming at connecting typical NC systems (UNIX-based) with dissimilar systems. For this purpose, the message-based GUTS operating system was stripped down to a modified version called *Nimrod*. Nimrod is an experimental attempt to building a heterogeneous NC environment. A central mapping agent, the so called NC process, provides message exchange between the UNIX and the Nimrod systems. Although the UNIX and the Nimrod systems are different, the hardware on which they run is not. The plan is to connect systems running on different hardware under the UNIXes-of-the-World-Unite philosophy.

Contact: C.R. Snow or H. Whitfield, Computing Laboratory, University of Newcastle upon Tyne, Claremont Road, Newcastle upon Tyne NE1 7RU, England.

References: [253], [254], [255], [81], [256], [257], [258], [259], [260]

2.38 NEXUS

Main Goal

The main purpose in building the NEXUS distributed operating system was to support experimental research in object-oriented distributed computing as well as fault-tolerance techniques. It provides access, location and failure transparency.

Advantages

Too much criticism is not appropriate since the system is not fully implemented and is only meant to be the basis for further experimental research.

Description

The system consists of a minimal distributed kernel which provides inter-object communication and a collection of objects. In the current design the kernel and each object are implemented as UNIX processes on a network of SUN workstations running UNIX 4.2 BSD. All UNIX functions are completely visible at the application level.

The NEXUS kernel provides synchronous and asynchronous communication based on the remote procedure call paradigm. All primitives use the datagram facility without any guarantee of delivery. If an exactly-once semantics is desired, transaction primitives must be used. If several requests are sent asynchronously, it is possible to wait for responses to them simultaneously.

Each object is controlled by an *object manager* that executes all operations, including concurrency control and recovery, on this object. Objects which have both the same set of operations and the same set of states are grouped into *classes*. UNIX processes managing these classes are called *class representatives*. Each representative manages a subset of the class and maintains a registry of probable representatives for each other object. Each object and each representative have a unique identifier (UID). An object can be located by its UID which contains the name of its representative or directly by the UID of the representative. The kernel maintains a cache to locate representatives and makes a broadcast if the cache does not contain an entry. A *CONTEXT* object provides a hierarchical name space and a *SHELL* object translates these symbolic object names in UIDs.

Security does not play any role.

A set of library functions for transaction management and concurrency control are provided, e. g. for creating two-phase-locking or commit protocols. These functions are supported by so called *virtual disk* storage objects. All invocations within a transaction are treated as nested transactions. NEXUS introduces the concept of *weak atomicity* meaning that atomicity of an action on an object is guaranteed only for changes made in this object's state.

Miscellaneous
Status: Currently, a prototype is being implemented in order to evaluate the system's structure and functionality.
Contact: Anand Tripathi, Dept. of Computer Science, University of Minnesota, Minneapolis, MN 55455
References: [261], [262]

2.39 NFS

Main Goal
The SUN Network Filesystem (NFS) was implemented to solve the problem of communication and sharing in a heterogeneous network of machines and operating systems by providing a shared file system location-transparently throughout the network.
Advantages
NFS is designed such that it can easily be transferred to other operating systems and machine architectures. It therefore uses an *External Data Representation* (XDR) specification. It describes protocols independently of machines and systems. In addition, the source code for user level implementation of the RPC and XDR libraries has been published. The main disadvantages of NFS are that it does not support all of the UNIX file system semantics. For example, due to a stateless protocol removing open files or file locking are not supported.
Description
NFS runs on various machine types and operating systems: UNIX 4.2 BSD, SUN OS, DEC Ultrix, System V.2, VMS and MS/DOS.
NFS is implemented on top of a Remote Procedure Call package in a client/server manner. NFS adds a new interface to the kernel. The client side is implemented inside the UNIX kernel. A mount service is implemented as an everlasting daemon process which is started every time a mount request is initialized. It is possible for any machine to be a client, a server or both.
The file system interface consists of two parts, first, the *Virtual File System* (VFS) interface which defines operations on the entire file system and second, the virtual node (*vnode*) interface which defines the operations on a single file within the file system.
Communication is synchronous and uses a stateless protocol. Because a server does not need to keep track of any past events, crash recovery is easy to implement. NFS uses the DARPA User Datagram Protocol (UDP) and the Internet Protocol (IP) for its transport level.
Performance of NFS: NFS uses normal cache-buffers (read-ahead, write-behind) to increase performance. For read-requests, performance is good enough to neglect local or remote access differences, but performance for write-requests is poor because it is synchronous at the server site.
Naming in NFS is done by using UNIX path names. The NFS servers export complete file systems. Clients can mount any file system branch of an exported remote file system on top of a local directory. Thus a hierarchical, uniform name space is presented to the users.
Security in NFS: The RPC package allows an encrypted authentication method, while still allowing access with a simpler authentication method for the PC-users. However, due to the distributed character of a network file system, the security features provided are weaker than those of the standard-UNIX. NFS uses UNIX-like permission checks between client and server sites. User-ID and group-ID are passed within the RPC and are used by the server to decide whether the client request for access can be satisfied or not. A server can easily protect its local file system by an export list of exported file systems and the authorized clients that can import each of them.
Failures: A server does not do any crash recovery at all. When a client crashes, no recovery is necessary (stateless protocol). If a server crashes the client resends requests until a response is received. NFS requests are idempotent. This stateless server implementation implies some UNIX incompatibilities. For example, if one client opens a file and another removes it, the first client's reads will fail even though the file is still open.
A useful extension to NFS is the so called *Yellow Pages* (YP) mechanism, i. e. a service that provides distribution

and central administration of read-only system databases. The exported file system information is maintained on each machine and made highly available this way.

Miscellaneous

Status and Contact: NFS is an established product and support on 4.2 BSD is available through SUN and Mt. Xinu, on system V through Lachman Associates and Unisoft, and on Ultrix through DEC (version 2.x).

The VMS implementation is for the server side only. The MS/DOS part is complicated by the need of a redirector layer that redirects system calls to the appropriate MS/DOS services.

References: [263], [264], [265], [266], [267], [268], [269], [270], [271], [272], [273], [50], [274], [275], [217]

2.40 Profemo

Main Goal

The main goal of the Profemo design is to provide nested transactions in a distributed object-oriented environment. Profemo guarantees location-, access- and concurrency-transparency.

Advantages

Profemo is an object-oriented approach resulting usually in poor performance. To compensate, inheritance is not defined on the language level, but is an integral feature of the Profemo design and supported by hardware.

Description

Profemo stands for "Design and Implementation of a fault tolerant multicomputer system" developed by the "Gesellschaft für Mathematik und Datenverarbeitung mbH" (GMD, West-Germany). Profemo is a new layer on top of UNIX 4.2 BSD. Profemo's architecture is divided into subsystems which communicate through messages via stream-sockets.

Profemo is based on an object model. Each object consists of three distinct parts, the object header containing semantic information about the object in order to achieve a well-defined interface, the capability part exclusively holding references to other objects, and the data part.

For some of the extended objects, a so called *invoke* function is defined which executes a type-specific operation providing simple synchronous calls and concurrent calls. Simple synchronous calls are RPC-based and use the Unreliable Datagram Protocol. Parameter passing is supplied by a parameter object which allows call by reference and call by value. Concurrent calls correspond to a traditional *fork*. As a result, the calling process is blocked until all results have been delivered indicating that all processes have been called terminated. Each parallel path of computation is a simple synchronous call in itself.

Transactions are the main form of computation. Communication between transactions is performed via shared data objects. A lock descriptor holds the information necessary for the non-strict two-phase locking protocol which guarantees the serializability of the operations performed on a shared object.

Capabilities are used for access control. Further sophisticated security issues have not been implemented.

As mentioned above, concurrency control is achieved by locking within a transaction scheme. Transactions can be nested.

Miscellaneous

Other issues: *Mutabor* (mapping unit for the access by object references) represent an efficient support meant to tailor a specific memory management unit. Mutabor provides efficient save/restore facilities and basic functions to manage the recovery points needed by the transaction scheme.

Status: Research is going on to look for alternative synchronization policies and to increase the system's performance.

Contact: E. Nett, Institut für Systemtechnik der Gesellschaft für Mathematik und Datenverarbeitung mbH, Schloss Birlinghoven, 5205 St. Augustin 1, West-Germany.

References: [276], [277], [278], [279], [280], [281]

2.41 PULSE

Main Goal

PULSE is a distributed operating system written partially in Ada. The main aspects of the project are the design of a distributed file system that could support standalone operation of workstations, as well as the evaluation of the Ada's suitability for distributed applications.

Advantages

The system provides a good environment for distributed Ada programs. The use of the prototype is circumstantial because two types of machines are required, one for developing and one for running applications.

Description

PULSE is being run on several LSI-11/23 that are connected by a Cambridge Ring. All development is being done on a VAX,i. e. cross-compiled and downloaded, since no Ada compiler exists for LSI. The kernel is written in C.
The PULSE IPC provides a virtual circuit and guarantees only that messages received will be in good condition and in sequence, but does not guarantee delivery. Messages can be sent synchronously, asynchronously or RPC-like.
The filesytem has a UNIX-like hierarchical structure. Files are kept in *volumes* that have unique numbers. Location is done by associated global *i-numbers*. Each file server maintains a list of locally on-line volumes. If a file is remote a request is broadcasted.
The protection scheme is the same as in UNIX.
Files and directories are replicated and updated with a *primary copy* mechanism. The duplicates are updated the next time they are referenced. For this purpose, version numbers are held. A user may control replication. When a client references a file, the file server attempts to find a copy in the following order: local master, local duplicate, remote master, remote duplicate.

Miscellaneous

Other issues: Several software tools, such as shell, ed, etc. are implemented in Ada.
Status: A prototype has been implemented. The current status of the project is unknown.
Contact: A.J. Wellings, Department of Computer Science, University of York, York, UK
References: [282], [283], [284], [285]

2.42 QuickSilver

Main Goal

QuickSilver developed at the IBM Almaden Research Center, is a client/server structured distributed system, which uses atomic transactions as a paradigm for recovery management. This way the system is access, location and failure transparent. Properly written programs should be resilient to external process and machine failures and should be able to recover all resources associated with failed entities.

Advantages

The system provides highly reliable services with little loss of performance, for example the recovery management overhead is low. On the other hand, nested transactions and language directed facilities are not available for defining and using recoverable objects.

Description

Quicksilver consists of a new operating system kernel, running on IBM RT-PCs inter-connected by an IBM token ring.
The inter-process communication can function synchronously, asynchronously (with a returned request-id for later using) or by way of messages. Most commonly, a request-response protocol is used, in which requests are neither lost nor duplicated, data is transferred reliably, and in case of a server crash, errors are reported to any clients involved (at-most-once semantics). Each service has a global unique address which is private or otherwise made public by registering with the name service.

Each resolution of a file name in the QuickSilver distributed file system is performed in the context of a specific user's file name space. The universe of users is partitioned into domains, each having a user location and identification service. User names are unique within a domain, so that the (domain, user name)-pair leads to a global unique name space. Each file server has a recoverable copy of all user indices and quotas. For the sake of performance, only whole files are transferred and are locally cached, by means of callbacks that assist in validating the caches. The only authentication is made at login. Afterwards, the kernel and the communication system are trusted as in a local system.
To cope with failures, each IPC, on behalf of a transaction, is tagged with a global unique transaction identifier. Transaction Managers (TM) are responsible for commit coordination by communicating with servers at its own node and TMs at other nodes. The commit coordinator can migrate to the subordinate or, in the case of more than one subordinate, be replicated to all subordinates. A Log Manager serves as a common recovery log both for the TM's commit and the server's recovery data. Servers are declared stateless, volatile or recoverable. No serialization is provided. This remains the responsibility of the servers managing the serializable resource.
Miscellaneous
Other issues and status: A Deadlock Detector is going to be developed that can detect global deadlocks and resolve them by aborting offending transactions. The system is running in daily production use at IBM Almaden. Experiments are being made with other applications, including a messaging facility and a mail store-and-forward system.
Contact: Roger Haskin, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120–6099
References: [286], [287]

2.43 RFS

Main goal
Remote File Sharing (RFS) is one of the networking-based features offered in the AT & T UNIX system V.3, providing location-transparent access to remote files and devices.
Advantages
RFS supports 100 % of the UNIX file system's semantics. This means that, in contrast to NFS (cf. section 2.39), file locking and append write are possible. Thus, existing user applications can use the features of RFS without modification or recompilation. The main disadvantages of RFS are, first, that only one of the UNIX versions (V.3) is supported, and second, that a new special protocol — non RPC-based — makes it hardly portable. Finally, there is no server crash recovery.
Description
RFS runs on multi-vendor hardware operating under UNIX System V.3 connected through an Ethernet or a Starlan LAN. RFS is based on a client/server model. Every machine can be a client, a server or both. RFS is a kernel extension that deals with file access requests as follows: The mount-service enables server machines to make remote files available to client machines. The RFS interface decides whether a request is local and can be satisfied at the local UNIX file system, or remote. If a request is remote communication is established via the Streams I/O system, developed for the UNIX time-sharing system for intermachine communication.
RFS has implemented a remote mount service allowing a user to add remote file system branches to its local file system tree. Subsequently, remote requests to these files are handled as if they resided inside the local machine. The notion of *domains* — an administrative name space made up of a collection of machines — was designed by RFS. For each domain, a name server exists. Name resolution is done by using this RFS name server which is modeled as a transaction handler: it receives requests, performs the operation desired, generates and sends back a reply to the originator. A so called *advertise* command gives a user the possibility to make subtrees of its local file system mountable for clients.
Communication between name servers on different machines is done by the standard Transport Layer Interface.

The TLI is based on the ISO Transport layer and is designed to support higher layer protocols in a protocol independent way. Both VC and datagram services are provided, enabling the use of different standard transport protocols (TCP/IP, ISO, XNS, SNA).
Concurrency control is achieved by file locking. The RFS uses a stateful protocol. Therefore, the server has to maintain information about the current state of all of its clients. Because it would be difficult and costly for the client to rebuild the server's state after a server crash, no server crash recovery is done.
Mapping of remote user-IDs or group-IDs to local user identifications is provided. This means, for example, that remote superusers can be mapped onto a local standard restricted user-ID.

Miscellaneous

Status and Contact: RFS is an integrated part of the commercial UNIX System V.3 developed by AT & T.
RFS allows access to remote special devices so that expensive peripheral devices can be shared economically.
References: [288], [289], [264], [290], [291], [292], [270], [50]

2.44 Saguaro

Main Goal

Saguaro is a distributed operating system which attempts to strike a balance between hiding the underlying hardware (e. g. providing location transparency) and allowing users to take advantage of it (e. g. placing replicas of files on different disks).

Advantages

Saguaro is a synthesis of new mechanisms and variations on ones from existing systems and provides various ways in which users can exploit the multiple machines offered to enhance performance and availability. It remains to be shown whether these concepts are feasible and perform well.

Description

A prototype implementation is being done on a network of eight SUN workstations. Portions of the file system are implemented on top of the UNIX kernel using the Ibis system (cf. section 2.29).
Processes communicate by synchronous calls or asynchronous send. Input and output of different commands can be connected by *channels* the functionality of which is a superset of UNIX pipes. Each channel has one write port and one or more read ports.
Files in Saguaro are organized into a single, hierarchical *Logical File System* (LFS) that is mapped onto one or more *Physical File Systems* (PFS). Any file or directory can be placed on any PFS, either explicitly by adding the PFS name or by default in the same PFS as its parent directory. Files are located by scanning of the path name. If a file along the path is inaccessible, a broadcast is made to the manager of every PFS which has access to the logical files stored on its PFS by a *virtual root*. In this way every file is accessible as long as the PFS on which it is stored is accessible.
Security has not been a research topic.
Files can be replicated by the use of special calls which have the file names of the replicas as parameters. The first call allows direct access to each replica and the second provides transparent access. The system tries to keep replicas consistent, returning error messages if this is not possible.

Miscellaneous

Other issues: In Saguaro, all data is typed and the system performs type checking on command invocation and file access. The basis for this is a data description language (UTS).
Servers are created dynamically and placed on an adequate host by a load manager. Physical resources are controlled by managers that are created during system initialization.
Status: Research is still going on.
Contact: Richard D. Schlichting, Department of Computer Science, University of Arizona, Tucson, AZ 85721
References: [293], [294], [295]

2.45 S-/F-UNIX

Main Goal

The main goal of this project at the University of Illinois in the early 80's was to extend the widely known and used UNIX time-sharing operating system to an access transparent distributed operating system. Performance of the communication software has been the major focus of research, while keeping most of the UNIX semantics.

Advantages

The project has shown that communication based on virtual circuits can achieve high transfer rates, fast response, and low local overhead costs. A disadvantage of S-/F-UNIX is its restriction to 64kB in regard to path names, read or written data. Terminal-to-terminal communication and interprocess pipes are not supported.

Description

S-/F-UNIX is a client/server system. *S-UNIX* is the specialized operating subsystem that runs user processes. The subsystem running on the file server is called *F-UNIX*. DEC PDP-11 line hosts communicate with each other through a high-bandwidth virtual circuit switch (VC). Small front-end processors handle the data and control protocol for error and flow-controlled virtual circuits. The conventional UNIX kernel was modified. Nodes are connected via the Datakit switch, which is a switch that combines the desirable properties of both packet and virtual circuit switching while still providing high and dynamically allocatable bandwidth, since it uses packet switching with demand multiplexing in its internal implementation. A so called NK (*network kernel*) protocol has been implemented to speed up transmission.

File space has been extended by mounting an entire disk volume on top of an existing directory. S-UNIX allows its user to mount a F-UNIX file server in an analogous way. Multiple mounts of both kinds can be active simultaneously. File names are hierarchical (path name strings). S-/F-UNIX uses remote inodes (*rinodes*) every time a remote file is accessed. It contains all the information needed for the S-UNIX subsystem to handle a specified file. This implies a pointer identifying the related F-UNIX machine holding the file and a unique number assigned by that F-UNIX machine. All other information about the file is maintained at the remote site.

Assuming that the operating system and the communication mechanisms are trustworthy, S-/F-UNIX avoids having to deal with problems of authentication beyond those present in the current UNIX system.

Miscellaneous

Other issues: Special files (devices) are treated like special files in UNIX. They are handled by dedicated F-UNIX subsystems.

Status: Not active.

Contact: G.W.R. Luderer, Bell Labs., Murray Hill, New Jersey 07974.

References: [296], [77], [297], [298], [299]

2.46 SMB

Main Goal

The Server Message Block (SMB) protocol was developed to support programmers who design or write programs that interact with the IBM PC Network Program via the IBM PC Network. Netbios and the Redirector, software elements of DOS 3.1, can be combined with SMB to provide most of the features found in distributed file systems. SMB provides access transparency.

Advantages

The main advantage of SMB is the fact that a huge user community — PC users under DOS — are enabled to take part in the world of distributed file systems. The final development of the features of DFS is still in progress and the disadvantage of not supporting crash recovery, transactions and security issues will be eventually eliminated due to a "critical mass" effect. The commercial character of this product and the number of installed systems involved will lead to a fast enhancement of SMB.

Description

SMB is the most visible methodology used in commercial networks, e. g. Microsoft's MS-Net and IBM's PC LAN Program. The native operating system of SMB is DOS (such as UNIX for NFS (cf. section 2.39) or RFS (cf. section 2.43)). SMB has thus far been implemented on UNIX V, UNIX 4.2 BSD and VMS, too. Applications running on a variety of PCs make standard system calls to DOS. These calls are intercepted by the so called *Redirector* which decides whether these calls require remote access or not. Given a remote access is necessary, the call is transferred to the Netbios software. Then Netbios establishes a reliable VC connection with a remote server. This server runs a special software which includes the share mechanism. Share is the main component of DOS 3.1. Share allows opening of remote files with various permissions. DOS semantics of the entire client file system is maintained.

The well known mounting mechanism allows expansion of a local file system by mounting remote file system branches on different file volumes. A hierarchical name space is the result.

SMB does not support any authentication mechanisms due to the fact that SMB was first created in a DOS environment in which security plays no role. However, authentication at the level of file system is possible by means of passwords for access to remote resources.

SMB uses a stateful protocol (like RFS). Detecting crashes is a problem and recovery mechanisms have not been defined.

File locking for concurrency control is provided.

Miscellaneous

Status and Contact: Microsoft's MS-Net and IBM's PC LAN Program are being marketed by Novel (Netware), 3COM (3+).

References: [300], [301], [270], [302]

2.47 SOS

Main Goal

The SOMIW (Secure Open Multimedia Integrated Workstation) Operating System (SOS) is a general-purpose distributed object-oriented operating system which provides various degrees of reliability. Each application only pays the price of those mechanisms it really needs and uses. It is a subtask of an Esprit project with the goal to construct an office workstation for manipulating, transporting and using multimedia documents. Access, location and failure transparency are provided.

Advantages

Due to its general approach, SOS is useful for many different application, but it remains to be proven whether these demands are met.

Description

The system is built upon a new minimal kernel completed with system services.

It provides a set of replaceable protocol objects built on top of a flexible transport protocol. In this way a reliable protocol can be chosen which detects and recovers lost messages, crashes and network failures and suppresses orphans. On the other hand, an unreliable protocol can be chosen as well. The maximum size of messages is given by the size of the network packet. On each host, there is one protocol manager for each type of protocol. On request it allocates a protocol object, connects it to the caller and to a protocol object on the callees host. This virtual connection is maintained indefinitely, or separately released by the protocol manager should the resource be needed. Normally, a RPC is used for client/server communication but message passing (including multicasting) is possible as well.

The only interface to a resource is via a *proxy*, the local representative of an object. A proxy is imported by a name service that has been preinstalled in every context. Each object has a global unique identifier.

Additionally, a proxy is an unforgettable capability that can make access-control application specific.

Protocol managers recover from a crash by sending a multicast to the other hosts to retrieve the lost state. Crashes of hosts are detected by probe messages which are sent out periodically.

Miscellaneous

Status: A prototype is currently being distributed to the partners of the Esprit project but performance is not being measured. Future research will include the specification of network interconnection and the exploration of application facilities.

Contact: Marc Shapiro, INRIA, B.P. 105, 78153 Le Chesnay Cédex, France

References: [303], [304], [305], [306], [307]

2.48 Sprite

Main Goal

The Sprite operating system was designed for a set of possible diskless workstations, connected by a local area network. The system is an extension of UNIX and is modeled as a set of services including a shared hierarchical file system and remote program execution with paging across the network. It provides access, location, replication and concurrency transparency.

Advantages

The Sprite file system has better performance than comparable file systems such as Andrew (cf. section 2.4) or NFS (cf. section 2.39) due to an efficient remote procedure call. In addition it copes better with concurrent writes. On the other hand, error recovery is poor, e. g. if a server crashes all clients need to be killed. The server utilization will probably become a bottleneck when the system grows. It is hardly portable because the communication protocol is only implemented for one type of local area network.

Description

Sprite is a new operating system kernel running on SUN workstations connected by an Ethernet and will be ported to SPUR, a multiprocessor workstation developed at Berkeley. The interface for user processes is much like that provided by UNIX. Remote links to files on hosts running UNIX are possible as well.

Its communication system is based mainly on an asymmetric kernel-to-kernel remote procedure call, which in turn is currently based on a special purpose protocol for an Ethernet. The internet protocol should be supported in the future. In order to reduce state information and allow implicit acknowledgements by getting replies or sending new requests, a set of clients is associated with one server process by the use of so called *client channels*. This way at-most-once semantics can be achieved by saving copies of the last reply message and use of sequence numbers as well as explicit acknowledgements and retransmissions in case of timeouts. A large message can be sent in one RPC by dividing it into fragments that allow partial retransmissions of lost fragments. Built-in delays that prevent overflowing by slow servers are not yet implemented. If a channel is idle, the server processes are freed by explicit end-of-channel transmissions. If there is no server available at the beginning of a transmission over a channel, the message is discarded. It is possible to broadcast an RPC, receiving only the first reply.

Sprite provides a hierarchical file system in which subtrees can be managed by different servers. Instead of a name service, each client has a prefix table which contains domain names, e. g. directories, the address of the server who manages this domain, and a short identifier of this directory. This table is constructed by using a broadcast and updated automatically as the configuration of servers and files changes. A server replies to open-file requests with the server-id and an identifier of the file or alternatively with the name of the server managing this file. The usage of working directories as well as the crossings of domains are problems.

The protection scheme is comparable to the one used in UNIX except that search permissions through a given path are granted to all users. Host-specific private files are available through the use of special table entries, refusing to answer lookup broadcasts, e. g. /usr/tmp.

Caching in main memory is needed for performance improvements, but is disabled in the case of concurrent writes. In addition, version numbers are held to change out-of-date cache blocks when files are sequentially write shared.

When a server crashes all associated client processes must be killed because of the loss of state information. If a server doesn't reply to an open request, the entry in the prefix table is invalidated and a new broadcast is made. This improves availability provided replicas exist on different servers. Servers can balance the load by offering different table entries to different clients, e. g. for system binaries. The system provides no atomic updates of replicas. Updates are not transparent because replicas have different prefixes.

Miscellaneous

Other issues: Processes can migrate to other nodes at any time. System calls, except calls related to the file system, are forwarded back to the home node of a process. If a process has a lot of dirty pages, the costs for migration are high. Paging can be done across the network. The amount of main memory used for virtual memory and for caching of files varies dynamically depending on demand.

Status: The system is still under development and a preliminary version is running on SUN-2 and SUN-3 workstations.

Contact: Fred Douglis or Michael N. Nelson, Computer Science Division, University of California at Berkeley, 571 Evans Hall, Berkeley, CA 94720.

For principal investigation, contact John Ousterhout at UC Berkeley.

References: [308], [309], [310], [311], [312], [313], [314]

2.49 SWALLOW

Main Goal

SWALLOW is a distributed file system supporting highly reliable object-oriented data storage. It provides all kind of transparency levels, i. e. those of location, access, concurrency, failure, and replication.

Advantages

SWALLOW is yet another object-oriented DFS. Each update creates a new version of the object. However, only the updated page is copied. All other pages are merely taken over by the new version. SWALLOW can manage objects of arbitrary size and provide synchronization and recovery mechanisms as part of the object model rather than on top of it.

Its main disadvantage is the handling of mass storage. Since every update operation creates a new version, a large amount of data needs to be stored. Performance has not yet been evaluated.

Description

SWALLOW runs on standard UNIX hardware in a client/server behavior. A so called *broker* software package provides access as well as location transparency for the client machines. Therefore, each client machine needs to run a broker copy.

SWALLOW is a distributed file system based on an object model. All data entities are encapsulated in objects.

SWALLOW stores its files in remote, highly autonomous servers called *repositories*. The core of a repository is a stable append-only storage called *Version Storage (VS)*. VS contains the version history of all data objects as well as information needed for crash recovery.

Every time an object is modified, a new version of the particular object is appended to the VS. Updates are embedded in transactions. In case of failures, object histories provide a backward recovery mechanism. Updates, which haven't been committed yet, are stored on stable storage.

SWALLOW uses an asynchronous datagram communication. Responses to requests are collected in an asynchronous fashion by the broker. A special end-to-end protocol (SWALLOW Message Protocol SMP) based on datagrams has been implemented for data transfer.

A capability-based protection scheme is implemented. Concurrency control — single writer, multiple reader — is achieved by timestamps. Each object is protected by a monitor providing mutual access control. The monitor state itself is kept in volatile memory and thus, after a crash, all objects are automatically unlocked.

In SWALLOW, data reliability is provided in a simple replication scheme. A duplicate is stored by a second storage device, and an update is applied immediately to both devices.

Miscellaneous

Status: SWALLOW is a data storage system designed in early 1980 at the Laboratory for Computer Science of the Massachusetts Institute of Technology. The main design components, repositories and brokers are prototypically implemented and tested under artificial workload. The current status of research is unknown.

Contact: Liba Svobodova, IBM Zurich Research Lab. 8803 Rüschlikon, Switzerland.

References: [315], [77], [316], [317], [82]

2.50 V

Main Goal

The V distributed system was developed at Stanford University with the purpose of connecting multiple workstations, access and location transparency provided. A small kernel is supposed to provide high performance communication and be used by servers which offer most other operating system facilities.

Advantages

The main advantage of V is its high-speed inter-process communication which is comparable to local communication. It also enables process groups (called teams) to be formed which communicate efficiently through multicasting. Unfortunately, no work has been done on reliability issues and failure handling.

Description

V is being run on VAXstationIIs, SUN-2/50s and SUN-3s. It consists of a small kernel which has to be booted over an Ethernet from a server on a machine running UNIX 4.x BSD. This way diskless workstations are supported. With a special server UNIX tools can be used, but the UNIX system must be modified in order to support multicasting and an Ethernet filter packet. Not all configurations are supported by the Stanford distribution tape.

V uses a special communication protocol, called Versatile Message Transaction Protocol (VMTP), which supports RPCs and client/server interactions. Messages contain a fixed-length message of 32 bytes and a 32-bit integer that uniquely identifies the destination process. Longer messages can be achieved by sending a pointer to a memory segment, where the first 1 kByte of the segment is *piggybacked* onto the message. A client looking for access to a server is blocked until the server has sent back a reply. The protocol provides an at-most-once semantics by using selective retransmissions and duplicate suppression. A special feature is its implementation of multicasting for group communication.

Each process has a unique process-id. Global unique names are achieved by adding to each object name the name of the server managing this object. If a name isn't known a multicast message is sent to all possible servers. To improve performance, (server name, pid) pairs are cached.

There are no special security issues, the same protection scheme is used as in UNIX.

As mentioned above, almost nothing has been done so far to improve reliability and availability of the system. There is only one exception server which gets a message whenever an exception occurs and which can thereafter invoke a debugger.

Miscellaneous

Other issues: Process migration is possible for process groups and can be used in a dynamic loadbalancing mechanism. To improve the performance of the migration mechanism, the address space of the process is precopied, so that only those pages referenced after this time need to be copied during the period of the real migration. All input and output operations use a uniform interface, called UIO, which is block-oriented.

Status: V is in use at several universities, research laboratories and companies. Further research is being done in porting V on a multiprocessor machine, connecting it to wide area networks and implementing reliability issues such as atomic transactions and replication.

Contact: David R. Cheriton or Tony Mason, Computer Science Department, Stanford University, Stanford CA 94305.
References: [318], [319], [320], [321], [322], [323], [324], [325], [326], [327], [328], [329], [330], [331], [332], [333], [146], [334], [29], [335], [336]

2.51 VAXcluster

Main Goal
The Digital Equipment Corporation has been developed the VAXcluster product as a set of closely-coupled DEC VAX computers providing a highly available and extensible configuration that operates access-, concurrency-, and failure-transparently. Another key concern was the performance of the system.
Advantages
The implementation of a high-speed message-oriented computer interconnect has increased the system's performance. In contrast to other highly available systems, VAXcluster is built from general-purpose, off-the-shelf processors, and a general-purpose operating system.
The main disadvantage of VAXcluster is that it does not provide location transparency.
Description
Each VAXcluster host runs a distributed version of the VAX/VMS operating system and is connected to the *CI* (Computer Interconnect) through a *CI port*. CI ports are device-specific and have been implemented so far for the VAX 11/750, 780, 782, 785, and VAX/8600 hosts. The CI logically is a bus. Physically, it is a dual path serial connection with each path supporting a 70 Mbit/second transfer rate.
The *SCA* (System Communication Architecture) is a software layer that provides datagram-style and VC communication. To ensure delivery of messages without duplication or loss, each CI port maintains a VC with every other cluster port. RPC-based communication is supported too.
The *Hierarchical Storage Controller* interprets a so called *Mass Storage Control Protocol (MSCP)* which separates the flow of control and status information from the flow of information to increase performance.
VAXcluster provides a cluster-wide shared file system to its users. A complete file name includes the disk device name (a system-wide unique name), the directory name, and the user-defined name itself. The disk device name gives the information concerning which node a particular file is located on. Therefore, location transparency is not achieved.
Password information resides in a single file shared by all VAXcluster nodes. A user obtains the same environment regardless of the node he is logged into.
A *lock manager* is the foundation of all resource sharing. It provides services for naming and locking of cluster-wide resources. A cluster connection manager is responsible for coordination within the cluster. It recognizes recoverable failures in remote nodes and provides data transfer that can handle such failures transparently.
To prevent partitioning, VAXcluster uses a quorum voting scheme. Initial vote and quorum values are set for each node by a system administrator.
Miscellaneous
Status and Contact: VAXcluster is a product still supported by several engineering groups at Digital Equipment Corporation.
References: [337]

2.52 XDFS

Main Goal
XDFS (Xerox Distributed File System) from Xerox PARC was a research project to design a very robust multiple-server system. It was intended to provide a basis for data base research. Besides, XDFS is location, access, and concurrency transparent.

Advantages
XDFS was an early approach in designing a distributed file system with sophisticated features such as stable storage, transaction handling, and concurrency control mechanisms. The main disadvantages of this system are the great overhead involved in the implementation of stable storage and the limited amount — a page — of data to be transferred as part of a single request.
Description
XDFS written in MESA language, runs on an Alto minicomputer, and communicates using an Ethernet communication system. Client communication in XDFS is built on the *Pup* (PARC Universal Packet) level, which implements an internetwork datagram.
The XDFS uses a B-tree to map file-ID and page number to a disk address. Thereby, access and location transparency are provided. It provides fine-grained locking at the byte level of a file.
Access control issues are simply based on the user-ID. This limits the degree of security provided.
For concurrency control (single writer, multiple reader), XDFS uses time-limited breakable locks. Deadlocks are detected through timeouts. All operations are repeatable. A two-phase commit protocol ensures correct updates. Failure handling and recovery are supported by shadow pages plus intention logs (redo log) which are stored at atomic stable storage. In XDFS, a client needs to set up a transaction before accessing or creating any file. All committed updates are completed immediately after a crash. If a new crash appears while the recovery procedure is active, the whole process will be restarted.
Replication is not supported.
Miscellaneous
Other issues: XDFS supports the database management system of Cedar (cf. section 2.9).
Status: The first version of XDFS became operational in 1977. The current status of the research is unknown.
Contact: J.G. Mitchell, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304
or J. Dion, Cambridge University Computer Lab., Corn Exchange Street, Cambridge, UK CB2 3QG.
References: [77], [80], [82]

3 Table of Comparison

The table of comparison is given to summarize and compare the systems discussed. It should be viewed carefully, since in certain ways any categorized comparison can be misleading. However, this way an easily legible overview may be obtained. The table provides quick access to a large amount of highly condensed information. The entries are organized according to the criteria used to describe the systems. Sometimes, a similar issue or a comparable feature for an entry have been implemented. We mark this with a special symbol (+).

Name	Type of System		Transparency				
	DOS	DFS	location	access	replication	concurrency	failure
Accent	*		*	*			*
Alpine		*		*			
Amoeba	*		*	*		*	*
Andrew		*	*	*	*	*	
Argus	*		*	*			*
Athena		+	+	+			
BirliX	*		*	*	*	*	*
Cambridge DS	*		*	*			*
Cedar		*	*	*	*	*	*
Charlotte	*		*	*			
Chorus	*		*	*			
Clouds	*		*	*		*	*
Cosmos	*		*	*	+	+	+
Cronus	*		*	*	+		
DACNOS	+		*	*		*	
DEMOS/MP	*		*	*			*
DOMAIN	*		*	*			
DUNIX	*		*	*			
Eden	*		*	*	*		*
EFS		*	*	*			
Emerald	+		*	*			
GAFFES		*	*	*	*	*	
Grapevine			*	*	*		
Guide	*		*	*		*	
Gutenberg	*		*	*	*	*	*
HARKYS		*	*	*			
HCS	+		*	*			
Helix		*	*	*			*
IBIS		*	*	*	*	*	
JASMIN	*		*				
LOCUS	*		*	*	*	*	*
Mach	*		*	*			*
Medusa	*		*	+			
Meglos	*		+	+			
MOS	*		*	*			
NCA/NCS	+		*	*	*	*	*
Newcastle		*	*	*			
NEXUS		*	*	*			*
NFS		*	*	*			
Profemo	*		*	*		*	
PULSE	*		*	*	*		
QuickSilver	*		*	*			*
RFS		*	*	*			
Saguaro	*		*	*	+		
S-/F-UNIX		*		*			
SMB		+		*			
SOS	*		*	*			*
Sprite	*		*	*	*	*	
SWALLOW		*	*	*	*	*	*
V	*		*	*			
VAXcluster		*		*		*	*
XDFS		*	*	*		*	*

Table 1: Table of comparison — Part 1

Name	Heterogeneity	
	OS	CPUs
Accent		PERQ
Alpine	(Cedar)	Dorado
Amoeba	UNIX 4.2 BSD	68000s, PDP11, VAXes, IBM-PC, NS32016
Andrew	UNIX 4.2 BSD	SUN 2/3, MVAXes, IBM RT PC
Argus	Ultrix 1.2	MVAXes
Athena	UNIX	multi vendor HW
BirliX	UNIX 4.3 BSD	
Cambridge DS	TRIPOS	LSI-4, 68000, Z-80, PDP-11/45
Cedar		Alto, Dorado WS
Charlotte	UNIX	VAX-11/750
Chorus	UNIX V 3.2	SM 90, IBM AT PC, 68000s, 80386
Clouds	UNIX	VAX-11/750, SUN 3/60
Cosmos	UNIX	
Cronus	UNIX V7, 4.2 BSD, VMS	68000s, VAXes, SUNs, BBN C70
DACNOS	VM/CMS, PC DOS, VMS	VAXes, IBM PC, IBM/370
DEMOS/MP		Z8000
DOMAIN	UNIX III, 4.2 BSD	Apollo
DUNIX	UNIX 4.1 BSD	VAXes
Eden	UNIX 4.2 BSD	SUN, VAXes
EFS	MASSCOMP RT UNIX	MASSCOMP MP
Emerald	Ultrix	MVAX II
GAFFES	all kind	all kind
Grapevine	multiple OS	multi vendor HW
Guide	UNIX V	Bull SPS 7/9, SUN 3
Gutenberg	UNIX	
HARKYS	multiple UNIX systems	multi vendor HW
HCS	multiple os	multi vendor HW
Helix	XMS	68010-based
IBIS	UNIX 4.2 BSD	VAX-11/780
JASMIN	UNIX V	
LOCUS	UNIX 4.2 BSD, V.2	DEC VAX/750, PDP-11/45, IBM PC
Mach	UNIX 4.3 BSD	all VAXes, SUN 3, IBM PC, NS32016
Medusa	StarOS	Cm*
Meglos	UNIX	68000s, VAXes, PM-68k
MOS	UNIX V7	PDP-11, PCS/CADMUS 9000
NCA/NCS	DOMAIN/IX, UNIX 4.x BSD, VAX/VMS	SUNs, IBM PCs, VAXes
Newcastle	UNIX V7, III, BSD, Guts	PDP-11
NEXUS	UNIX 4.2 BSD	SUN
NFS	UNIX 4.2 BSD, V.2, VMS, SUN OS, Ultrix, MS/DOS	multi vendor HW
Profemo	UNIX 4.2 BSD	DEC VAXes
PULSE		LSI-11/23
QuickSilver		IBM RT-PC
RFS	UNIX V.3	multi vendor HW running UNIX V.3
Saguaro		SUN
S-/F-UNIX	UNIX V	DEC PDP-11
SMB	UNIX V, UNIX 4.x BSD, VMS;DOS 3.x	multi vendor PC
SOS		
Sprite	UNIX 4.x BSD	SUN 2/3
SWALLOW		
V	UNIX 4.x BSD	SUN 2/3, VAXstationII
VAXcluster	VAX/VMS	VAX 7xx-11s, VAX/8600
XDFS		Alto

Table 2: Table of comparison — Part 2

Name	Changes made		Communication			RPC-	Connection		
	new kernel	new layer	standard protocols	specialized protocols	shared memory	based	VC	datagram	pipes/ streams
Accent	*			*				*	
Alpine	*					*			
Amoeba	*			Amoeba		*			
Andrew	*		UDP/IP			*		*	
Argus		*		ACP/IP				*	
Athena		+	TCP/IP				*		
BirliX	*				*	*		*	
Cambridge DS	*			*		*		*	*
Cedar		*	FTP				*		
Charlotte	*		*				*	*	
Chorus	*		*			*		*	
Clouds	*				*	*			
Cosmos	*								
Cronus	*		TCP, UDP	VLN		*	*	*	
DACNOS		*	OSI			*		*	
DEMOS/MP	*			*		*		*	
DOMAIN	*			*	*				
DUNIX	*		TCP/IP	*				*	
Eden		*				*			
EFS	*			RDP		*			
Emerald		*							
GAFFES		+	*			*	*		
Grapevine			*					*	*
Guide		*			*				
Gutenberg		*				*			
HARKYS		*	*			*			
HCS		*	*			*			
Helix	*		OSI				*		
IBIS		*	TCP/IP			*	*		
JASMIN	*			Paths					
LOCUS	*						*		
Mach	*			*	*	*		*	
Medusa	*				+		*		*
Meglos	*			*			*		
MOS	*		UDP/IP			*		*	
NCA/NCS		*	UDP/IP			*		*	
Newcastle		*				*			
NEXUS		*	*			*		*	
NFS	*		UDP/IP			*		*	
Profemo		*	UDP/IP		*	*			*
PULSE	*					*	*		
QuickSilver	*			*		*		*	
RFS	*		OSI TLI						*
Saguaro									*
S-/F-UNIX	*			KP			*		
SMB	*			SMB			*		
SOS	*			*		*		*	
Sprite	*			*		*			
SWALLOW		*		SMP				*	
V	*			VMTP		*		*	
VAXcluster	*			MSCP		*	*	*	
XDFS		*		Pup				*	

Table 3: Table of comparison — Part 3

Name	Semantics			Naming		Security			
	may be	at most once	exactly once	object-oriented	hier-archical	en-cryption	special HW	capa-bilities	mutual auth.
Accent		*		*				*	
Alpine		*			*	*		*	*
Amoeba		*		*		*	*	*	
Andrew	*				*				*
Argus			*	*					
Athena					*				
BirliX	*			*	*			*	
Cambridge DS	*	*		*				*	
Cedar		*			*				
Charlotte	*				*			*	
Chorus				*					
Clouds			*	*					
Cosmos		*		*				+	
Cronus	*			*				+	
DACNOS		*		*					*
DEMOS/MP	*		*		*			+	
DOMAIN				*	*				
DUNIX	*				*				
Eden		*		*				*	
EFS	*				*				
Emerald	*			*					
GAFFES			*		*	*	*	*	*
Grapevine					*				
Guide		*		*					
Gutenberg			*	*				*	
HARKYS	*				*				
HCS									
Helix				*		*		*	
IBIS	*				*				*
JASMIN	*				*			*	
LOCUS		*	*		*				
Mach		*		*				*	
Medusa				*					
Meglos					*				
MOS	*				*				
NCA/NCS		*		*					
Newcastle	*	*			*				
NEXUS	*		*	*	*				
NFS	*				*	*			
Profemo				*				*	
PULSE				*					
QuickSilver		*	*		*				
RFS	*				*				
Saguaro					*				
S-/F-UNIX	*				*				
SMB	*				*				
SOS	*	*		*				*	
Sprite		*			*				
SWALLOW		*		*				*	
V		*			*				
VAXcluster	*				*				
XDFS		*			*				

Table 4: Table of comparison — Part 4

Name	Availability				Failures				Process Migration
	synchro-nization	TA	nested TA	repli-cation	recovery client crash	recovery server crash	stable storage	orphan detection	
Accent					+	+			*
Alpine	+	*			+	+			
Amoeba	*					*	*	*	
Andrew	*			*					
Argus	*	*	*	*	*	*	*	*	
Athena									
BirliX	*			*	*	*		*	*
Cambridge DS		+				*		+	
Cedar	*	*		*	*	*			
Charlotte						*			*
Chorus									*
Clouds	*	*	*		*	*	*		
Cosmos	*	*		*					
Cronus				*					*
DACNOS	*								
DEMOS/MP					*	*			*
DOMAIN									
DUNIX									
Eden		*		*		*			*
EFS		*			*	*			
Emerald	+								*
GAFFES	*	*	*	*					
Grapevine				*					
Guide	+	*	*						
Gutenberg	*	*		*			*		
HARKYS									
HCS									
Helix		*	*		*			+	
IBIS	*			*	*	*			
JASMIN									
LOCUS	*	*	*	*	*	*			*
Mach					+	+			*
Medusa	*								*
Meglos	+								
MOS				*					*
NCA/NCS	+			*					
Newcastle									
NEXUS		*	*						
NFS					*	*			
Profemo	*	*	*						
PULSE	*			*					
QuickSilver		*				*	*		
RFS	*				*				
Saguaro				*					
S-/F-UNIX									
SMB									
SOS					*	*			
Sprite	*			*					*
SWALLOW	*	*		*	*	*	*		
V									*
VAXcluster	*				*	*			
XDFS	*	*			*	*	*		

Table 5: Table of comparison — Part 5

4 Related Systems

In this section we present a list of related systems which aren't described in the survey. The list contains a short description of and a main reference to each system.

Acorn

The Acorn File Server developed at Acorn Computers ltd. supports diskless workstations in schools by providing a shared hierarchical file system. The Acorn File Server ran on M 6502 with floppy disks.

Reference: [338] ([82])

Arachne

Arachne is the direct predecessor of the Charlotte operating system (the Crystal project, cf. section 2.10). Arachne is a distributed operating system kernel that was designed at the Computer Sciences Department of the University of Wisconsin–Madison.

Reference: [339] ([340])

Arca

Arca is a centralized file system server developed at the Department of Computing of the University of Lancaster, Bailrigg, Lancaster, LA1 4YR. Arca is based on the Cambridge File Server (cf. section 2.8) but, unlike CFS which runs on an operating system kernel called TRIPOS, Arca is written as a monolithic program on a bare machine.

Arca serves two different LANs: a Ethernet-type network called *Strathnet*, and a Cambridge Ring-type network called *Polynet*. VAX–11/750s, PDP–11s, and M68000 hosts are attached to these networks. Two access protocols are being used by Arca: the Single-Shot Protocol and the RPC. An atomic update mechanism is implemented. Ports, represented by a 16-bit integer, are known addresses to which requests and replies can be sent.

In future, it is planned to integrate stable storage into Arca, and to distribute the file server among two or more machines.

Reference: [341]

ArchOS

Archons is a large and long-term project at Carnegie-Mellon University performing research on *decentralized computers*, including the design of special-purpose OS support hardware. ArchOS is a system-wide, but physically replicated, operating system that manages all the global resources through teams which negotiate, compromise, and reach a best-effort consensus based on inaccurate and incomplete information. This is supported by a general transaction facility.

Reference: [342]

Camelot

In the Camelot project a distributed transaction facility combined with high performance have been constructed. It executes on a variety of uni- and multi-processors on top of Mach whose description can be found in the survey (cf. section 2.32).

Reference: [343]

CICS

CICS (Customer Information Control System) is a commercial product of the IBM Corp. designed as a transaction processing system. CICS provides transaction services to resource managers such as DL/1, System 2000, and System R. It also provides a record interface to terminals and to sessions through virtual circuits. CICS is access transparent using a classic RPC facility. The RPC works for queues, processes, files, DL/1 databases, System 2000 databases, and other objects. CICS maintains a system-wide log for transactions and uses the standard two-phase commit protocol to coordinate the transaction commit and to deal with node or link failures. Replication transparency is not supported. Global deadlocks are detected by timeouts.

CICS implements IBM's SNA (System Network Architecture) and provides communication with non-CICS and non-IBM systems via a well-defined interface.

Reference: [344]

Circus
Circus is a replicated remote procedure call facility. The so called *paired message protocol* of Circus is based on the connectionless DARPA UDP. It is responsible for the segmentation of messages which are longer than a single datagram. Multiple segments are allowed to be sent out before one has been acknowledged. It is currently implemented in user code under UNIX 4.2 BSD.
Reference: [345] or [346]

Clearinghouse
The commercial product Clearinghouse is a distributed and replicated name server. It is developed from Xerox Corp. and is a successor to Grapevine (cf. section 2.23), but provides a hierarchical name space with three levels and uses different replication algorithms. Currently, about 350 Clearinghouse services are in operation all over the world. The main problem resulting from this size is the long propagation of updates (days or weeks) and the heavy Clearinghouse-induced internetwork traffic.
Reference: [347]

CMCFS
CMCFS (Carnegie-Mellon Central File System) was designed in early 1980. It provides two kinds of update schemes. First, the immutable file approach by which a new version is created with each update and, second, a traditional update based on transactions. CMCFS is part of the Spice project.
Reference: [348] ([82])

Cocanet
Cocanet is a local computer network based on the UNIX operating system. It extends the existing file system name space by adding the host name to the resource name, e.g. /vax1/usr/bin. Additionally, standard creation primitives can be used to execute remote programs.
Reference: [349]

Coda
Coda is a resilient distributed file system, currently being developed at Carnegie-Mellon-University. It should provide failure transparency.
Reference: [350] (see also [48])

CONIC
The CONIC environment is designed to support the construction and operation of software for distributed embedded systems. It employs a host/target approach, providing a comprehensive set of tools for program compilation, building, debugging and execution on the host, and supports distributed operation on the targets. It is being developed at the Department of Computing, Imperial College, London.
Reference: [351]

DASH
Dash is a distributed system for wide area networks from the University of Berkeley. At the heart of DASH is the authenticated database protocol (ADP) which is based on public key encryption. Above ADP there is a session-oriented resource access structure akin to UNIX streams. Filter processes in the stream provide a reliable transmission.
Reference: [352, 353]

Datacomputer
The Datacomputer is a data management and storage utility developed to share resources in the Arpanet. Clients are normal Arpanet hosts. The Datacomputer was implemented on DEC PDP-10 and offered services on the Arpanet in late 1973.
Reference: [354] ([82])

DEMOS

DEMOS is an operating system for a Cray 1. Many of the facilities are being used in the distributed version DEMOS/MP which is described in the survey (cf. section 2.16).

Reference: [355]

DFS925

DFS925 is a Distributed File System for the workstations 925. DFS925 has been developed at the Office Systems Lab. at the IBM San Jose Research Lab., Ca. with the goal of building an integrated system providing location transparency and atomic transactions. Replication of files is not supported.

Reference: [356]

DISTOS

DISTOS is a distributed operating system at the University of Kaiserslautern, W. Germany. The operating system functions are decomposed into multiple distribution units assigned to computers attached to a high-speed LAN. Part of the project was the development of CSSA, a language for programming distributed applications.

Reference: [357]

DISTRIX

DISTRIX is a message-based implementation of UNIX System V for a network of workstations developed by Convergent Technologies, USA. The kernel is a collection of server processes based on a real-time operating system called the *Foundation System*. Processes communicate either synchronously or asynchronously in a location transparent fashion.

Reference: [358]

Dragon Slayer

Dragon Slayer is a distributed operating system developed at the Computer Science Dept. of the Wayne State University, Detroit, MI 48202.

A distributed resource scheduling algorithm implemented makes Dragon Slayer a special development providing for services for novel applications, e. g. in office information systems.

Reference: [359]

DUNE

DUNE is a distributed operating system that provides uniform services across various network types. The services include remote file access, process migration and load balancing. The inter-process communication is based on an extended RPC, called *service request*, that permits device and application specific optimizations. DUNE is developed at Bellcore, Morristown, NJ 07960.

Reference: [360]

Enchère

Enchère is a distributed electronic marketing system consisting of loosely-coupled workstations that communicate via messages. Reliability is guaranteed by means of transactions that are supported by new hardware and software products. Stable memory boards were developed to guarantee the reliability of transactions. The orientation of current research is to extend Enchère to a general-purpose, object-oriented system.

Reference: [361]

Encompass

Encompass is a distributed database management system for the TANDEM computers. It is built on top of a recovery mechanism based on replication of both hardware and software.

Reference: [362] ([82])

Felix

Felix is a file server for a distributed system (i.e. Helix, cf. section 2.28) developed at Bell-Northern Research. It supports virtual memory and sharing of data and provides secure access by the use of capabilities.

Reference: [197]

Firefly
Firefly is a shared-memory multiprocessor workstation of DEC Systems Research Center that contains from one to seven processors. The processors have coherent caches, so that each sees a consistent view of main memory. See Topaz for description of the software system.
Reference: [363]
Galaxy
Galaxy is a distributed operating system designed at the university of Tokyo. It provides network-wide shared memory access.
Reference: [364]
GFS
The Generic File System for UNIX has been created by Digital Equipment Corporation to support all types of hierarchical file systems (local, remote, stateful, stateless), e.g NFS (cf. section 2.39), Berkeley FFS or MS-DOS file system. GFS controls all common file system resources and becomes the sole interface for file system operations.
Reference: [365]
IDRPS
The Intelligent Distributed Resource Processing System (IDRPS) of the Software Productivity Consortium should provide transparent access to a wide variety of resources without any concern for access techniques. The system will utilize as many components of existing systems as possible. NFS (cf. section 2.39) is chosen to provide remote file access. Transparent remote process execution can be achieved by adding a layer of software on top of NCS (cf. section 2.36). Some features like authentication and load balancing should be added. The environment includes Apollo, VAX, Gould, TI Explorers, SUN and Symbolic AI machines.
Reference: [366]
IFS
IFS (Interim File System) developed at Xerox PARC is a distributed file system organized in a tree of directories. IFS is operational since 1975 in connection with Alto microcomputers. It is written in the BCPL language. IFS provides a repository of shared files and is used for backup and archiving of private files.
Reference: [367] ([82])
ISIS
The ISIS system provides tools for creating and managing process groups, group broadcast, failure detection and recovery, distributed execution and synchronization, etc. It is based on virtual synchrony achieved by a reliable broadcast mechanism. The initial version of ISIS is being run on UNIX on SUN, DEC, GOULD and HP systems.
Reference: [368]
MANDIS
The MANDIS project explores management issues in an environment of local and wide area networks. It provides transparent communication in an internetwork and tools for managing a distributed system. A prototype which is based on Amoeba (cf. section 2.3) has been installed in four European countries.
Reference: [369]
MICROS
MICROS is the distributed operating system for the MICRONET network computer. It is intended for control of network computers of thousands of nodes. A main design goal is the optimization of network costs, such as internode communication and processing throughput. MICRONET is a network of 16 loosely-coupled LSI–11 microcomputer nodes, connected by packet-switching interfaces to pairs of medium-speed shared communication busses (500 kB/second). Each node runs a private copy of the MICROS kernel written in Concurrent PASCAL. All network processes communicate via an uniform message passing system. For large networks, MICROS resources are managed in nested pools. Each management node controls all resources within its subtree of the network hierarchy.

Addressable entities have globally unique names consisting of three fields informing of its type, the creating node, and a unique number within that node.
In future MICROS is intended to provide a viable system for computer science research and possible commercial development.
Reference: [370]

Nest

The Nest project copes with the problems arising in an extended distributed UNIX environment. It is a kernel level implementation of UNIX. The kernel code of a few system calls was changed preserving the system call interface. Nest provides location independent remote execution, process migration, and load balancing.
A prototype implementation uses a communication subsystem which is similar to the one-paths used in the DEMOS and Roscoe projects (cf. this section). A so called *server pool* scheme has been designed. Here, each workstation maintains a local database of servers which it is interested in in order to get CPU power as well as a database of clients to which it will offer some CPU cycles.
Reference: [371]

NonStop

The Tandem NonStop System is a distributed computer system designed expressly for on-line transaction processing. Fault-tolerance is achieved by pairs of processes doing the same work on different processors. Processes communicate by messages.
Reference: [372]

R$\star$

R$\star$ is a distributed database management system. Multisite user transactions are structured in a *tree* of processes communicating over virtual circuit communication links. Beside the transaction management, distributed deadlock detection protocols are implemented. R$\star$ is running on multiple processors.
Reference: [373]

RIG

Rochester's Intelligent Gateway had provided access to Arpanet services and resources as well as remote file access and other network facilities available on a local Ethernet, until its demise in 1986 because its hardware base became obsolescent. RIG is a predecessor of Accent (cf. section 2.1).
Reference: [1]

Roscoe

Roscoe is a multi-computer distributed operating system running on a network of DEC LSI–11 machines. Roscoe was developed in 1977 and 1978 at the Computer Sciences Department of Wisconsin-Madison. There is a direct line between research on Roscoe and Arachne and the Charlotte system (cf. section 2.10).
Reference: [374] ([375], [376], [377])

RSS

RSS was developed at the IBM San Jose Research Lab. as part of the system R project. RSS is the layer of the system R which is responsible for transaction management and database recovery.
Reference: [378] ([82])

RT PC Distributed Services

RT PC Distributed Services provides distributed operating system capabilities for the AIX operating system. These include a distributed file system and distributed interprocess communication. Transparency is achieved by *remote mounts* of files or file systems. Most application programs are running without modification.
Reference: [379]

S/Net's Linda Kernel

Linda is a parallel programming language which supports shared-memory-like distributed data structures. S/Net is a multicomputer based on a fast word-parallel bus interconnect. A special communication kernel is implemented

for this machine that supports Linda primitives (cf. section 2.34).
Reference: [238]

Sesame

Sesame is a file system, developed at Carnegie-Mellon-University as part of the Spice project. It is a successor to CMCFS (cf. this section). Its main objectives are naming, authentication, authorization and data storage as well as retrieval in a network of personal computers.
Reference: [380]

StarOS

StarOS was the first operating system developed for the Cm* multimicroprocessor computer. Cm* was designed at the Carnegie-Mellon University, Pittsburg, PA. The successor to StarOS is Medusa (cf. section 2.33).
Reference: [381]

STORK

STORK is an experimental migrating file system for computer networks developed at the Department of Computer Science of the Purdue University, West Lafayette. STORK is a direct predecessor of the IBIS file system (cf. section 2.29).
Reference: [201]

Thoth

Thoth is a real-time operating system which is designed to be portable over a large set of machines. Therefore, it is written in a high-level language. It provides efficient interprocess communication primitives that influenced the development of V (cf. section 2.50).
Reference: [382]

Topaz

Topaz is the software system for the Firefly multiprocessor (cf. this section). The key facilities of Topaz are execution of existing Ultrix binaries, a remote file system (Taos), a display manager and a debug server (TTD). Topaz provides light-weighted processes and remote procedure calls for inter-address-space and inter-machine communications. The performance of the RPC is high even though relatively slow processors are used.
Reference: [383]

TILDE

TILDE is a design project at the Department of Computer Science of the Purdue University, West Lafayette, to build a Transparent Integrated Local and Distributed Environment (TILDE). Part of the TILDE project is the IBIS file system (cf. section 2.29).
Reference: [200]

TRFS

The Transparent Remote File System (TRFS) provides transparent access in a distributed UNIX file system through the use of symbolic links to path names on remote machines, e.g. /libc/libc.a → /@mach1/libc/libc.a. Thus, programs don't need to be recompiled or relinked. TRFS is a project of Integrated Solutions, Inc.
Reference: [384]

Trollius

Trollius is an INMOS transputer (hypercube) mulitasking, operating system originally implemented for the FPS T-series. It is a direct successor to *Trillium*.

Trollius combines a transputer, one or more Unix 4.3 BSD hosts and any other Unix-based machine into one large system. Trollius does not include implicit support for parallel processing by itself, but instead provides explicit control of all facets of machine operation including reconfiguration of the operating system. Trollius uses a simple client/server model to enable every process in the multicomputer to transparently access a UNIX-compatible filesystem. Location transparency is not provided, since filenames are prefixed, if necessary, with a node-id. OSI-like datalink, network and transport services are provided.

The developers at The Ohio State University Office of Research Computing, Columbus Ohio, and at the Cornell Theory Center, Ithaca NY., intend to operate in a mode similar to Open Software Foundation.
Reference: [385]

UPPER

UPPER (Universal Poly-Processor with Enhanced Reliability) is an object-oriented multicomputer system of GMD FIRST, Germany. The operating system is supported by special hardware. All communication is done by asynchronous client-server interactions. Server processes are replicated to achieve fault-tolerance.
Reference: [386]

WFS

WFS (Woodstock File Server) is an experimental file server from Xerox PARC supporting the Woodstock office system. WFS has been operational since 1975 and provides sharing of information in applications, such as an experimental telephone directory, and a library information storage and retrieval system.
Reference: [387] ([82])

Xcode

The Xcode is a distributed system designed and implemented at the Politecnico di Milano within the Cnet project. It runs on Olivetti M 40s connected by an Ethernet and provides location transparency to all private resources. Xcode consists of an abstract machine layer *Virtual nodes* and a runtime support layer.
Reference: [388]

Z-Ring File Server

The Z-Ring File Server was developed at the IBM Zurich Research Lab. to build a testbed for the then experimental token ring network. It provides a hierarchical file system. The Z-Ring File Server was implemented in BCPL under the TRIPOS operating system.
Reference: [389] ([82])

Acknowledgements

We are pleased to acknowledge Kathy Schlag and Ernst Biersack for their helpful comments and careful reading of earlier versions of the paper.

References

[1] J.E. Ball, J.R. Feldman, J.R. Low, R.F. Rashid, and P.D. Rovner, "RIG, Rochester's Intelligent Gateway: System Overview", *IEEE Transactions on Software Engineering*, SE–2(4):321–328, December 1976.

[2] R. Fitzgerald and R.F. Rashid, "The Integration of Virtual Memory Management and Interprocess Communication in Accent", *ACM Transactions on Computer Systems*, 4(2):147–177, May 1986.

[3] R.F. Rashid, "An Inter-Process Communication Facility for UNIX", Technical Report CMU–CS–80–124, Departement of Computer Science, Carnegie Mellon Univ., February 1980.

[4] R.F. Rashid and G.G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel", In *Proc. 8th Symp. on Operating Systems Principles*, pages 64–75, August 1981.

[5] R.F. Rashid, "The Accent Kernel Interface Manual", Technical Report, Departement of Computer Science, Carnegie Mellon Univ., January 1983.

[6] R.F. Rashid, "Experiences with the Accent Network Operating System", In *Proc. Int. Seminar on Networking in Open Systems*, pages 270–295, LNCS #248, Springer Verlag, August 1986.

[7] E. Zayas, "Attacking the Process Migration Bottleneck", In *Proc. 11th Symp. on Operating Systems Principles*, pages 13–24, 1987.

[8] M.R. Brown, K. Kolling, and E.A. Taft, "The Alpine File System", Technical Report CSL–84–4, Xerox PARC, Palo Alto, Ca., October 1984, also in ACM Transactions on Computer Systems 3(4):261–293, Nov. 1985.

[9] M.R. Brown, K.N. Kolling, and E.A. Taft, "The Alpine File System", *ACM Transactions on Computer Systems*, 3(4):261–293, November 1985.

[10] E. H. Baalbergen, "Parallel and Distributed Compilations in Loosely-Coupled Systems", In *Proc. Workshop "Large Grain Parallelism"*, Providence, RI, October 1986.

[11] H.E. Bal, A.S. Tanenbaum, J.M. van Staveren, and J. Hall, "A Distributed, Parallel, Fault Tolerant Computing System", Informatica Report IR–106, Dept. of Mathematics and Computer Science, Vrije Univ., Amsterdam, October 1985.

[12] H.E. Bal and R. van Renesse, "Parallel Alpha-Beta Search", In *Proc. NGI-SION Symp. Stimulerende Informatica*, pages 379–385, Utrecht, Netherlands, April 1986.

[13] S.J. Mullender and A.S. Tanenbaum, "Protection and Resource Control in Distributed Operating Systems", *Computer Networks*, 8(5,6):421–432, 1984.

[14] S.J. Mullender, "Distributed Systems Management in Wide-Area Networks", In *Proc. NGI/SION Symp.*, pages 415–424, Amsterdam, April 1984.

[15] S.J. Mullender and A.S. Tanenbaum, "Immediate Files", *Software — Practice and Experience*, 14(4):365–368, April 1984.

[16] S.J. Mullender and R. van Renesse, "A Secure High-Speed Transaction Protocol", In *Proc. Cambridge EUUG Conf.*, September 1984.

[17] S.J. Mullender and P.M.B. Vitanyi, "Distributed Match-Making for Processes in Computer Networks", In *Proc. 4th ACM Principles of Distributed Computing*, Minaki, Canada, Aug 1985.

[18] S.J. Mullender and A.S. Tanenbaum, "A Distributed File Service Based on Optimistic Concurrency Control", In *Proc. 10th Symp. on Operating Systems Principles*, pages 51–62, Orcas Island, Washington, December 1985.

[19] S.J. Mullender and R. van Renesse, "A Secure High-Speed Transaction Protocol", In *Proc. Cambridge EUUG Conf.*, pages 125–136, September 1984.

[20] S.J. Mullender and A.S. Tanenbaum, "The Design of a Capability-Based Distributed Operating System", *The Computer Journal*, 29(4):289–300, March 1986.

[21] S.J. Mullender, "Making Amoeba work", In *2nd SIGOPS Workshop "Making Distributed Systems work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[22] S.J. Mullender, "Interprocess Communication", In *Arctic 88, An Advanced Course on Distributed Systems*, chapter 3, Tromsø, Norway, July 1988.

[23] S.J. Mullender, "Protection", In *Arctic 88, An Advanced Course on Distributed Systems*, chapter 4, Tromsø, Norway, July 1988.

[24] S.J. Mullender, G. van Rossum, A.S. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba: A Distributed Operating System for the 1990s", *IEEE Computer*, 23(5):44–53, May 1990.

[25] R. van Renesse, A.S. Tanenbaum, and S.J. Mullender, "Connecting UNIX Systems Using a Token Ring", In *Proc. Cambridge EUUG Conf.*, September 1984.

[26] R. van Renesse, "From UNIX to a Usable Distributed Operating System", In *Proc. EUUG Autumn '86 Conf.*, pages 15–21, Manchester, UK, September 1986.

[27] R. van Renesse, A.S. Tanenbaum, J.M. van Staveren, and J. Hall, "Connecting RPC-Based Distributed Systems Using Wide-Area Networks", Informatica Report IR–118, Dept. of Mathematics and Compute Science, Vrije Univ., Amsterdam, December 1986.

[28] A.S. Tanenbaum, S.J. Mullender, and R. van Renesse, "Capability-Based Protection in Distributed Operating Systems", In *Proc. Symp. Certificering van Software*, Utrecht, Netherlands, November 1984.

[29] A.S. Tanenbaum and R. van Renesse, "Distributed Operating Systems", *ACM Computing Surveys*, 17(4):419–470, December 1985.

[30] A.S. Tanenbaum and R. van Renesse, "Making Distributed Systems Palatable", In *2nd SIGOPS Workshop "Making Distributed Systems work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[31] A.S. Tanenbaum, S.J. Mullender, and R. van Renesse, "Using Sparse Capabilities in a Distributed Operating Systems", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 558–563, May 1986.

[32] A.S. Tanenbaum and R. van Renesse, "Reliability Issues in Distributed Operating Systems", In *Proc. 6th Symp. Reliability of Distributed Software & Database Systems*, pages 3–11, Williamsburg, Virginia, March 1987.

[33] M. Accetta, G. Robertson, M. Satyanarayanan, and M. Thompson, "The Design of a network-based central file system", Technical Report CMU–CS–80–134, Dept. of Computer Science, Carnegie-Mellon Univ., August 1980.

[34] N. Borenstein, C. Everhart, J. Rosenberg, and A. Stooler, "A Multi-media Message System for Andrew", In *Proc. Usenix Conf.*, 1988.

[35] R.B. Dannenberg, *Resource Sharing in a Network of Personal Computers*, PhD thesis, Carnegie-Mellon Univ., December 1982.

[36] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, "Scale and Performance in a Distributed File System", In *Proc. 11th Symp. on Operating Systems Principles*, pages 1–2, November 1987.

[37] J.H. Howard, M.J. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[38] J. Leong, "Data Communication at CMU", Technical Report CMU–ITC–85–043, Carnegie-Mellon Univ., July 1985.

[39] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment", *Communications of the ACM*, 29(3):184–201, March 1986.

[40] J.H. Morris, ""Make or Take" Decisions in Andrew", In *Proc. Usenix Conf.*, 1988.

[41] D.A. Nichols, "Using Idle Workstations in a Shared Computing Environment", In *Proc. 11th Symp. on Operating Systems Principles*, pages 5–12, November 1987.

[42] A.J. Palay, W.J. Hansen, M.L. Kazar, M. Sherman, M.G. Wadlow, T.P. Neuendorffer, Z. Stern, M. Bader, and T. Peters, "The Andrew Toolkit — An Overview", In *Proc. Usenix Conf.*, 1988.

[43] M. Satyanarayanan, "Supporting IBM PCs in a Vice/Virtue Environment", Technical Report CMU-ITC-002, Information Technology Center, Carnegie-Mellon Univ., 1984.

[44] M. Satyanarayanan, J.H. Howard, D.A. Nichols, R.N. Sidebotham, A.Z. Spector, and M.J. West, "The ITC Distributed File System: Principles and Design", In *Proc. 10th Symp. on Operating Systems Principles*, pages 35–50, ACM SIGOPS, December 1985.

[45] M. Satyanarayanan, "Integrating Security in a Large Distributed Environment", Technical Report CMU–CS–87–179, Departement of Computer Science, Carnegie-Mellon Univ., 1987.

[46] M. Satyanarayanan, "Distributed file systems", In *Arctic 88, An Advanced Course on Distributed Systems*, chapter 6, Tromsø, Norway, July 1988.

[47] M. Satyanarayanan, "On the Influence of Scale in a Distributed System", In *Proc. 10th Int. Conf. on Software Engineering*, 1988.

[48] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access", *IEEE Computer*, 23(5):9–22, May 1990.

[49] B. Sidebotham, "VOLUMES — The Andrew File System Data Structure Primitive", *Proc. EUUG Autumn '86*, pages 473–480, September 1986.

[50] G. Vandôme, "Comparative Study of some UNIX Distributed File Systems", In *Proc. EUUG Autumn '86*, pages 73–82, Manchester, September 1986.

[51] M. West and D. Nichols, "The ITC Distributed File System: Prototype and Experience", Technical Report CMU–ITC–040, Carnegie-Mellon Univ., March 1985.

[52] I. Greif, R. Seliger, and W. Weihl, "A Case Study of CES: A Distributed Collaborative Editing System Implementation in Argus", *Programming Methodology Group Memo 55, MIT Lab for Comp. Science, Cambridge, Ma.*, April 1987.

[53] R. Ladin, B. Liskov, and L. Shrira, "A Technique for Constructing Highly Available Services", *Algorithmica*, 3:393–420, 1988.

[54] B. Liskov, A. Snyder, R.R. Atkinson, and J.C. Schaffert, "Abstraction mechanisms in CLU", *Communications of the ACM*, 20(8):564–576, August 1977.

[55] B. Liskov, "The Argus Language and System", In M. Paul und H.J. Siegert, editor, *Distributed Systems — Methods and Tools for Specification*, pages 343–430, LNCS #190, Springer Verlag, 1982.

[56] B. Liskov and R.W. Scheifler, "Guardians and Actions: Linguistic Support for Robust, Distributed Programs", *ACM Transactions on Programming Languages and Systems*, 5(3):381–404, July 1983.

[57] B. Liskov, "Overview of the Argus language and system", *Programming Methodology Group Memo 40, MITambridge, Ma.*, February 1984.

[58] B. Liskov and R. Ladin, "Highly-Available Distributed Services and Fault-Tolerant Distributed Garbage Collection", In *Proc. 5th Symp. on Principles of Distributed Computing*, pages 29–39, August 1986.

[59] B. Liskov, D. Curtis, P. Johnson, and R. Scheifler, "Implementation of Argus", *Proc. 11th Symp. on Operating Systems Principles*, 21(5):111–122, November 1987.

[60] B. Liskov, R. Scheifler, E. Walker, and W. Weihl, "Orphan Detection", *Programming Methodology Group Memo 53, MIT Lab for Computer Science, Cambridge, Ma.*, July 1987.

[61] B. Liskov and al., "Argus Reference Manual", Technical Report MIT/LCS/TR–400, MIT Lab for Computer Science, Cambridge, Ma., 1987.

[62] B. Liskov, "Distributed Programming in Argus", *Communications of the ACM*, 31(3):300–312, March 1988.

[63] E.W. Walker, "Orphan Detection in the Argus System", Technical Report MIT/LCS/TR 326, MIT Lab for Computer Science, Cambridge, Ma., June 1984.

[64] W. Weihl and B. Liskov, "Implementation of Resilient, Atomic Data Types", *ACM Transactions on Programming Languages and Systems*, 7(2):244–269, April 1985.

[65] E. Balkovich, S. Lerman, and R.P. Parmelee, "Computing in Higher Education: The Athena Experience", *IEEE Computer*, 18(11):112–125, November 1985.

[66] E. Balkovich, S. Lerman, and R.P. Parmelee, "Computing in Higher Education: The Athena Experience", *Communications of the ACM*, 28(11):1214–1224, November 1985.

[67] J. Gettys, "Project Athena", In *Proc. Usenix Summer '84 Conf.*, pages 72–77, Salt Lake City, June 1984.

[68] R.J. Sonza and S.P. Miller, "UNIX and Remote Procedure Calls: A Peaceful Coexistence?", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 268–277, Cambridge, MA., May 1986.

[69] H. Härtig, W. Kühnhauser, W. Lux, H. Streich, and G. Goos, "Structure of the BirliX Operating System", In *Proc. EUUG Autumn '86*, pages 433–449, Manchester, UK, September 1986.

[70] H. Härtig, W. Kühnhauser, W. Lux, and H. Streich, "Distribution in the BirliX Operating System", In *Proc. NTG/GI-Fachtagung Architektur und Betrieb von Rechensystemen*, pages 345–357, Stuttgart, West Germany, March 1986.

[71] H. Härtig, W. Kühnhauser, W. Lux, H. Streich, and G. Goos, "Distribution and Recovery in the BirliX Operating System", In N. Gerner and O. Spaniol, editors, *Kommunikation in verteilten Systemen*, Informatik-Fachberichte #130, pages 190–201, Springer Verlag, February 1987.

[72] H. Härtig, W. Kühnhauser, W. Lux, W. Reck, and H. Streich, "The BirliX Operating System", Internal Report, Gesellschaft für Mathematik und Datenverarbeitung mbH, GMD, St. Augustin, West Germany, June 1988.

[73] O.C. Kowalski and H. Härtig, "Protection in the Birlix Operating System", In *Proc. 10th Int. Conf. on Distributed Computing Systems*, pages 160–166, Paris, France, May 1990.

[74] J. Dion, "The Cambridge File Server", *Operating Systems Review*, 14(4):26–35, October 1980.

[75] J. Dion, "Reliable Storage in a Local Network", Technical Report 16, Univ. of Cambridge Computer Laboratory, 1981.

[76] N.H. Garnett and R.M. Needham, "An Asynchronous Garbage Collector for the Cambridge File Server", *Operating Systems Review*, 14(4):36–40, October 1980.

[77] A. Hac, "Distributed File Systems — A Survey", *Operating Systems Review*, 19(1):15–18, January 1985.

[78] A.J. Herbert and R.M. Needham, "The User Interface to the Cambridge Distributed System", In *Proc. 8th Symp. on Operating Systems Principles*, December 1981.

[79] A. Hopper, "The Cambridge Ring — A Local Network", In F.K. Hanna, editor, *Advanced Techniques for Microprocessor Systems*, 1980.

[80] J.G. Mitchell and J. Dion, "A Comparison of Two Network-Based File Servers", *Communications of the ACM*, 25(4):233–245, April 1982.

[81] R.M. Needham and A.J. Herbert, *The Cambridge Distributed System*, Addison Wesley, 1982.

[82] L. Svobodova, "File Servers for Network-Based Distributed Systems", *ACM Computing Surveys*, 16(4):353–398, December 1984.

[83] M.V. Wilkes and R.M. Needham, "The Cambridge Model Distributed System", *Operating Systems Review*, 14(1):21–29, January 1980.

[84] J. Donahue, "Integration Mechanisms in Cedar", *ACM SIGPLAN Notices*, 20(7):245–251, July 1985.

[85] D.K. Gifford, R.M. Needham, and M.D. Schroeder, "The Cedar File System", *Communications of the ACM*, 31(3):288–298, March 1988.

[86] R. Hagmann, "Reimplementing the Cedar File System Using Logging and Group Commit", In *Proc. 11th Symp. on Operating Systems Principles*, pages 155–162, November 1987.

[87] E.E. Schmidt, "Controlling Large Software Development in a Distributed Environment", Technical Report CLS–82–7, Xerox Palo Alto Research Center, Ca., December 1982.

[88] M.D. Schroeder, D.K. Gifford, and R.M. Needham, "A Cashing File System for a Programmer's Workstation", Technical Report CLS–85–7, Xerox Palo Alto Research Center, Ca., November 1985.

[89] D.C. Swinehart, P.T. Zellweger, and R.B. Hagmann, "The Structure of Cedar", *ACM SIGPLAN Notices*, 20(7):230–244, July 1985.

[90] D. Swinehart and al., "A Structural View of the Cedar Programming Environment", *ACM Transactions on Programming Languages and Systems*, 8(4):419–490, October 1986.

[91] C. Thacher and al., "Alto: A Personal Computer", Technical Report CLS–79–11, Xerox Palo Alto Research Center, Ca., August 1979.

[92] W. Teitelman, "The Cedar Programming Environment: A Midterm Report and Examination", Technical Report CLS–83–11, Xerox Palo Alto Research Center, Ca., July 1984.

[93] R.A. Finkel, M.H. Solomon, D. DeWitt, and L. Landweber, "The Charlotte Distributed Operating System", Technical Report 502, Univ. of Wisconsin–Madison Computer Sciences, October 1983.

[94] F. Armand, M. Gien, M. Guillemont, and P. Leonard, "Towards a Distributed UNIX System – The CHORUS Approach", *Proc. EUUG Autumn '86*, pages 413–431, September 1986.

[95] J.S. Banino, A. Caristan, and H. Zimmermann, "Synchronization for Distributed Systems using a single Broadcast Channel", In *Proc. 1st Int. Conf. on Distributed Computing Systems*, 1979.

[96] J.S. Banino, A. Caristan, M. Guillemont, G. Morisset, and H. Zimmermann, "CHORUS: An Architecture for Distributed Systems", Technical Report 42, INRIA, France, November 1980.

[97] J.S. Banino and J.C. Fabre, "Distributed Coupled Actors: A CHORUS Proposal for Reliability", In *Proc. 3rd Int. Conf. on Distributed Computing Systems*, page 7, Ft. Lauderdale, Fl., USA, October 1982.

[98] J.S. Banino, J.C. Fabre, M. Guillemont, G. Morisset, and M. Rozier, "Some Fault Tolerant Aspects of the CHORUS Distributed System", In *Proc. 5th Int. Conf. on Distributed Computing Systems*, Denver, Colorado, May 1985.

[99] J.S. Banino, G. Morisset, and M. Rozier, "Controlling Distributed Processing with CHORUS Activity Messages", In *18th Hawaii Int. Conf. on System Science*, January 1985.

[100] J. Banino, "Parallelism and Fault-Tolerance in the CHORUS Distributed System", In *Int. Workshop on Modelling and Performance Evaluation of Parallel Systems*, Grenoble, December 1984.

[101] M. Guillemont, "The CHORUS Distributed Computing System: Design and Implementation", In *Int. Symp. on Local Computer Networks*, pages 207–223, Florence, April 1982.

[102] F. Herrmann, F. Armand, M. Rosier, M. Gien, V. Abrossimov, I. Boule, M. Guillemont, P. Léonard, S. Langlois, and W. Neuhauser, "Chorus, a new Technology for Building UNIX-Systems", In *Proc. EUUG Autumn '88*, pages 1–18, Cascais, Portugal, October 1988.

[103] M. Rozier and J. M. Legatheaux, "The Chorus Distributed Operating System: Some Design Issues", In Y. Paker et al., editor, *Distributed Operating Systems: Theory and Practice*, volume F28 of *NATO ASI series*, pages 261–289, Springer Verlag, August 1986.

[104] H. Zimmermann, J.S. Banino, A. Caristan, M. Guillemont, and G. Morisset, "Basic Concepts for the support of Distributed Systems: The CHORUS approach", In *Proc. 2nd Int. Conf. on Distributed Computing Systems*, pages 60–66, 1981.

[105] H. Zimmermann, M. Guillemont, G. Morisset, and J.S. Banino, "CHORUS: A Communication and Processing Architecture for Distributed Systems", Technical Report 328, INRIA Research, September 1984.

[106] M. Ahamad and P. Dasgupta, "Parallel Execution Threads: An Approach to Atomic Actions", Technical Report GIT-ICS-87/16, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[107] M. Ahamad, M.H. Ammar, J. Bernabeau, and M.Y.A. Khalidi, "A Multicast Scheme for Locating Objects in a Distributed Operating System", Technical Report GIT–ICS–87/01, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[108] J.E. Allchin, "An Architecture for Reliable Decentralized Systems", Technical Report GIT-ICS-83/23, School of Information and Computer Science, Georgia Institute of Technology, Ph.D. thesis, 1983.

[109] J.M. Bernabeau-Auban, P.W. Hutto, and M.Y.A. Khalidi, "The Architecture of the Ra Kernel", Technical Report GIT-ICS-87/35, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[110] J. Bernabéau-Aubán, P. Hutto, M. Yousef, A. Khalidi, M. Ahamad, W. Appelbe, P. Dasgupta, R. LeBlanc, and U. Ramachandran, "Clouds — A Distributed, Object-Based Operating System, Architecture and Kernel Implementation", In *Proc. EUUG Autumn '88*, pages 25–37, Cascais, Portugal, October 1988.

[111] R. Chen and P. Dasgupta, "Consistency-Preserving Threads: Yet Another Approach to Atomic Programming", Technical Report GIT-ICS-87/43, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[112] P. Dasgupta, R. LeBlanc, and E. Spafford, "The Clouds Project: Design and Implementation of a Fault-Tolerant Distributed Operating System", Technical Report GIT-ICS-85/29, School of Information and Computer Science, Georgia Institute of Technology, 1985.

[113] P. Dasgupta, R.J. LeBlanc, and W.F. Appelbe, "The Clouds Distributed Operating System", In *Proc. 8th Int. Conf. on Distributed Computing Systems*, pages 1–9, San Jose, Ca., June 1988.

[114] E.H. Spafford, "Kernel Structures for a Distributed Operating System", Technical Report GIT–ICS–86/16, School of Information and Computer Science, Georgia Institute of Technology, Ph.D. thesis, 1986.

[115] D.V. Pitts and P. Dasgupta, "Object Memory and Storage Management in the Clouds Kernel", In *Proc. 8th Int. Conf. on Distributed Computing Systems*, pages 10–17, San Jose, Ca., June 1988.

[116] C.T. Wilkes, "Preliminary Aeolus Reference Manual", Technical Report GIT-ICS-85/07, School of Information and Computer Science, Georgia Institute of Technology, 1985.

[117] C.T. Wilkes and R.J. LeBlanc, "Rationale for the Design of Aeolus: A Systems Programming Language for an Action/Object System", Technical Report GIT–ICS–86/12, School of Information and Computer Science, Georgia Institute of Technology, 1986.

[118] C.T. Wilkes, "Programming Methodologies for Resilience and Availability", Technical Report GIT–ICS–87/32, School of Information and Computer Science, Georgia Institute of Technology, Ph.D. thesis, 1987.

[119] G.S. Blair, J.R. Nicol, and C.K. Yip, "A Functional Model of Distributed Computing", Technical Report CS-DC-1–86, Dept. of Computing, Univ. of Lancaster, Lancaster, U. K., 1986.

[120] G.S. Blair, J.A. Mariani, and J.R. Nicol, "COSMOS — A Nucleus for a Program Support Environment", Technical Report CS–SE–1–86, Dept. of Computing, Univ. of Lancaster, Lancaster, U. K., 1986.

[121] G.S. Blair, J.R. Nicol, and J. Walpole, "An Overview of the COSMOS Distributed Programming Environment Project", Technical Report CS–DC–4–87, Dept. of Computing, Univ. of Lancaster, Lancaster, U. K., 1987.

[122] J.R. Nicol, G. Blair, and J. Walpole, "Operating System Design: Towards a Holistic Approach?", *Operating Systems Review*, 21(1):11–19, January 1987.

[123] M.A. Dean, R.E. Sands, and R.E. Schantz, "Canonical Data Representation in the Cronus Distributed Operating System", In *Proc. IEEE INFOCOM '87*, pages 814–819, San Francisco, Ca., April 1987.

[124] R.F. Gurwitz, M.A. Dean, and R.E. Schantz, "Programming Support in the Cronus Distributed Operating System", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 486–493, Cambridge, MA, May 1986.

[125] R.E. Schantz, R.H. Thomas, and G. Bono, "The Architecture of the Cronus Distributed Operating System", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 250–259, Cambridge, MA, May 1986.

[126] H. v. Drachenfels, "Ein Orientierungsdienst im Informationsverarbeitungsverbund", In *NTG/GI-Fachtagung Architektur und Betrieb von Rechensystemen*, pages 321–331, Stuttgart, West Germany, March 1986.

[127] H. Eberle and H. Schmutz, "NOS Kernels for Heterogeneous Environments", In *Proc. Int. Seminar on Networking in Open Systems*, pages 270–295, LNCS #248, Springer Verlag, 1986.

[128] H. Eberle, K. Geihs, A. Schill, H. Schmutz, and B. Schöner, "Generic Support for Distributed Processing in Heterogeneous Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 80–109, Springer Verlag, 1988.

[129] C. Förster, "Task Setup Service for Distributed Systems", In N. Gerner and O. Spaniol, editors, *Kommunikation in Verteilten Systemen*, Informatik-Fachberichte #130, pages 154–166, Springer Verlag, February 1987.

[130] C. Förster, "Controlling Distributed User Tasks in Heterogeneous Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 183–199, Springer Verlag, 1988.

[131] K. Geihs and M. Seifert, "Validation of a Protocol for Application Layer Services", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 306–320, Springer Verlag, 1988.

[132] K. Geihs and M. Seifert, "Automated Validation of a Co-operation Protocol for Distributed Systems", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 436–443, Cambridge, MA, May 1986.

[133] K. Geihs, R. Staroste, and H. Eberle, "Operating System Support for Heterogeneous Distributed Systems", In N. Gerner and O. Spaniol, editors, *Kommunikation in Verteilten Systemen*, Informatik-Fachberichte #130, pages 178–189, Springer Verlag, February 1987.

[134] U. Hollberg and E. Krämer, "Transparent Access to Remote Files in Heterogeneous Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 140–168, Springer Verlag, 1988.

[135] B. Mattes and H. v. Drachenfels, "Directories and Orientation in Heterogeneous Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 110–125, Springer Verlag, 1988.

[136] B. Mattes, "Authentication and Authorization in Resource Sharing Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 126–139, Springer Verlag, 1988.

[137] R. Öchsle, "A Remote Execution Service in a Heterogeneous Network", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 169–182, Springer Verlag, 1988.

[138] M. Seifert and H. Eberle, "Remote Service Call: A Network Operating System Kernel for Heterogeneous Distributed Systems", In *NTG/GI-Fachtagung Architektur und Betrieb von Rechensystemen*, pages 292–305, Stuttgart, West Germany, March 1986.

[139] M. Seifert and H. Eberle, "Remote Service Call: A NOS Kernel and its Protocols", In *Proc. 8th ICCC*, pages 675–680, September 1986.

[140] R. Staroste, H. Schmutz, M. Wasmund, A. Schill, and W. Stoll, "A Portability Environment for Communication Software", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 51–79, Springer Verlag, 1988.

[141] H. Wettstein, H. Schmutz, and O. Drobnik, "Cooperative Processing in Heterogeneous Computer Networks", In G. Krüger and G. Müller, editors, *HECTOR*, volume II: Basic Projects, pages 32–50, Springer Verlag, 1988.

[142] B.P. Miller, D.L. Presotto, and M.L. Powell, "DEMOS/MP: The Development of a Distributed System", *Software — Practice and Experience*, 17(4):277–290, April 1987.

[143] M.L. Powell, "The DEMOS File System", In *Proc. 6th Symp. on Operating Systems Principles*, pages 33–42, Purdue Univ., November 1977.

[144] M.L. Powell and B.P. Miller, "Process Migration in DEMOS/MP", In *Proc. 9th Symp. on Operating Systems Principles*, pages 110–119, Bretton Woods, N.H., October 1983.

[145] M.L. Powell and D.L. Presotto, "PUBLISHING: A Reliable Broadcast Communication Mechanism", In *Proc. 9th Symp. on Operating Systems Principles*, pages 100–109, Bretton Woods, N.H., October 1983.

[146] E.D. Lazowska, J. Zahorjan, D.R. Cheriton, and W. Zwaenepooel, "File Access Performance of Diskless Workstations", Technical Report 84–06–01, Dept. of Computer Science, Univ. of Washington, June 1984.

[147] P.J. Leach, B.L. Stumpf, J.A. Hamilton, and P.H. Levine, "UID's as Internal Names in a Distributed File System", In *Proc. 1st Symp. on Principles of Distributed Computing*, pages 34–41, Ottawa, 1982.

[148] P.J. Leach, P.H. Levine, B.P. Douros, J.A. Hamilton, D.L. Nelson, and B.L. Stumpf, "The Architecture of an Integrated Local Network", *IEEE Journal on Selected Areas in Communications*, SAC–1(5):842–857, November 1983.

[149] P.J. Leach, P.H. Levine, J.A. Hamilton, and B.L. Stumpf, "The File System of an Integrated Local Network", In *Proc. ACM Computer Science Conf.*, pages 309–324, New Orleans, March 1985.

[150] D.B. Leblang and R.P. Chase, "Computer-Aided Software Engineering in a Distributed Workstation Environment", In *Proc. SIGSOFT/SIGPLAN Software Engineering Symp. on Practical Software Environments*, pages 104–112, 1984.

[151] P.H. Levine, "The Apollo DOMAIN Distributed File System", In Y. Paker et al., editor, *Distributed Operating Systems: Theory and Practice*, volume F28 of *NATO ASI series*, pages 241–260, Springer Verlag, August 1986.

[152] P. Levine, "The DOMAIN System", In *2nd ACM SIGOPS Workshop on "Making Distributed Systems Work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[153] D.L. Nelson and P.J. Leach, "The Evolution of the Apollo DOMAIN", In *Proc. COMPCON Spring '84*, pages 132–141, San Francisco, Ca., February 1984, also in Proc. HICSS 1984, pages 470–479.

[154] D.L. Nelson and P.J. Leach, "The Architecture and Applications of the Apollo DOMAIN", *IEEE Computer Graphics and Applications*, April 1984.

[155] J. Rees, P.H. Levine, N. Mischkin, and P.J. Leach, "An Extensible I/O System", In *Proc. Usenix Summer '86 Conf.*, pages 114–125, Atlanta, June 1986.

[156] J. Rees, M. Olson, and J. Sasidhar, "A Dynamically Extensible Streams Implementation", In *Proc. Usenix Summer '87 Conf.*, pages 199–207, 1987.

[157] A. Litman, "DUNIX — A Distributed UNIX System", In *Proc. EUUG Autumn '86*, pages 23–31, Manchester, UK, September 1986, also in Operating Systems Review, 22(1):42–50, January 1988.

[158] G. Almes, A. Black, C. Bunje, and D. Wiebe, "Edmas: A Locally Distributed Mail System", In *Proc. 7th Int. Conf. on Software Engineering*, pages 56–66, March 1984.

[159] G.T. Almes, A.P. Black, E.D. Lazowska, and J.D. Noe, "The Eden System: A Technical Review", *IEEE Transactions on Software Engineering*, SE–11(1):43–58, January 1985.

[160] S.A. Banawan, "An Evaluation of Load Sharing in Locally Distributed Systems", Technical Report 87-08-02, Dept. of Computer Science FR-35, Univ. of Washington, Seattle, August 1987.

[161] A.P. Black, "An Asymetric Stream Communication System", In *Proc. 9th Symp. on Operating Systems Principles*, pages 4–10, October 1983.

[162] A.P. Black, N. Hutchinson, B.C. McCord, and R.K. Raj, "EPL Programmers Guide, Version 3.0", Technical Report, Eden Project, Dept. of Computer Science, Univ. of Washington, June 1984.

[163] A. Black, "Supporting Distributed Applications", *Operating Systems Review*, 19(5):181–193, December 1985.

[164] W.H. Jessop, D.M. Jacobson, J.D. Noe, J.-L. Baer, and C. Pu, "The Eden Transaction Based File System", In *Proc. 2nd Symp. on Reliability in Distributed Software and Database Systems*, pages 163–169, July 1982.

[165] E.D. Lazowska, H.M. Levy, G.T. Almes, M.J. Fischer, R.J. Fowler, and S.C. Festal, "The Architecture of the Eden System", In *Proc. 8th Symp. on Operating Systems Principles*, pages 148–159, December 1981.

[166] J. Noe, A. Proudfoot, and C. Pu, "Replication in Distributed Systems: The Eden Experience", In *Proc. Fall Joint Computer Conf.*, pages 1197–1210, Dallas, Texas, November 1986.

[167] C. Pu, J. Noe, and A. Proudfoot, "Regeneration of Replicated Objects: A Technique and its Eden Implementation", In *Proc. 2nd Int. Conf. on Data Engineering*, pages 175–187, February 1986.

[168] C.T. Cole, P.B. Flinn, and A.B. Atlas, "An Implementation of an Extended File System for UNIX", In *Proc. Usenix Summer '85 Conf.*, pages 131–149, Portland, June 1985.

[169] A. Black, N. Hutchinson, E. Jul, and H. Levy, "Object Structure in the Emerald System", In *Proc. 1st ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, SIGPLAN Notices*, volume 21,11, pages 78–86, November 1986.

[170] A. Black, N. Hutchinson, E. Jul, H. Levy, and L. Carter, "Distribution and Abstract Types in Emerald", *IEEE Transactions on Software Engineering*, SE–13(1):65–76, January 1987.

[171] S. Gozani, M. Gray, S. Keshav, V. Madisetti, E. Munson, M. Rosenblum, S. Schoettler, M. Sullivan, and D. Terry, "GAFFES: The Design of a Globally Distributed File System", Technical Report UCB/CSD 87/361, Computer Science Division (EECS), Univ. of California, Berkeley, June 1987.

[172] D.B. Terry, "Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments", Technical Report UCB/CSD 85/228, Computer Science Division Univ. of Berkeley, Ca., March 1985.

[173] A.D. Birrell, R. Levin, R.M. Needham, and M.D. Schroeder, "Grapevine: An Exercise in Distributed Computing", *Communications of the ACM*, 25(4):260–274, April 1982.

[174] M.D. Schroeder, A.D. Birrell, and R.M. Needham, "Experience with Grapevine: The Growth of a Distributed System", *ACM Transactions on Computer Systems*, 2(1):3–23, February 1984.

[175] R. Balter, A. Donnelly, E. Finn, C. Horn, and G. Vandôme, "Distributed Systems for Local Area Networks", Technical Report, IMAG Campus, Grenoble Cédex, France, August 1986.

[176] R. Balter, D. Decouchant, A. Freyssinet, S. Krakowiak, M. Meysembourg, C. Roisin, X. Rousset de Pina, R. Scioville, and G. Vandôme, "Guide: An Object-Oriented Distributed Operating System", Technical Report, IMAG Campus, Grenoble Cédex, France, 1988.

[177] D. Decouchant, A. Duda, A. Freyssinet, E. Paire, M. Riveill, X. Rousset de Pina, and G. Vandôme, "Guide, An Implementation of the COMANDOS Object-Oriented Distributed System Architecture on UNIX", In *Proc. EUUG Autumn '88*, pages 181–193, Cascais, Portugal, October 1988.

[178] S. Krakowiak, "Guide", In *2nd ACM SIGOPS European Workshop on "Making Distributed Systems Work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[179] S. Krakowiak, M. Meysembourg, M. Riveill, and C. Roisin, "Design and Implementation of an Object-Oriented Strongly Typed Language for Distributed Applications", Technical Report, IMAG Campus, Grenoble Cédex, France, September 1987.

[180] P. Chrysanthis, K. Ramamritham, and D. Stemple, "The Gutenberg Operating System Kernel", In *Proc. Fall Joint Computer Conf.*, pages 169–177, Dallas, Texas, November 1986.

[181] K. Ramamritham, D. Stemple, and S.T. Vinter, "Primitives for Accessing Protected Objects", In *Proc. 3rd Symp. on Reliability in Distributed Software and Database Systems*, pages 114–121, Clearwater Beach, October 1983.

[182] K. Ramamritham, D. Stemple, and S.T. Vinter, "Decentralized Access in a Distributed System", In *Proc. 5th Int. Conf. on Distributed Computing Systems*, pages 524–532, Denver, Colorado, May 1985.

[183] K. Ramamritham, D. Stemple, D. Briggs, and S. Vinter, "Privilege Transfer and Revocation in a Port-Based System", *IEEE Transactions on Software Engineering*, SE–12(5):635–648, May 1986.

[184] D. Stemple, K. Ramamritham, S.T. Vinter, and T. Sheard, "Operating System Support for Abstract Database Types", In *Proc. 2nd Int. Conf. Databases*, pages 179–195, September 1983.

[185] D.W. Stemple, S.T. Vinter, and K. Ramamritham, "Functional Addressing in Gutenberg: Interprocess Communication without Process Identifiers", *IEEE Transactions on Software Engineering*, SE–12(11):1056–1066, November 1986.

[186] S. Vinter, K. Ramamritham, and D. Stemple, "Protecting Objects through the Use of Ports", In *Proc. Phoenix Conf. Computers and Communication*, pages 399–404, March 1983.

[187] S. Vinter, *A Protection Oriented Distributed Kernel*, PhD thesis, Dept. of Computing and Information Sciences, Univ. Massachusetts, September 1985.

[188] S.T. Vinter, K. Ramamritham, and D. Stemple, "Recoverable Actions in Gutenberg", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 242–249, Cambridge, MA., May 1986.

[189] A. Baldi and L. Stefanelli, "HARKYS — A New Network File System Approach", In *Proc. EUUG Autumn '86*, pages 163–173, Manchester, UK, September 1986.

[190] B.N. Bershad, D.T Ching, E.D. Lazowska, J. Sanislo, and M. Schwartz, "A Remote Procedure Call Facility for Heterogenous Computer Systems", *IEEE Transactions on Software Engineering*, SE–13(8), August 1987.

[191] B.N. Bershad and H.N. Levy, "A Remote Computation Facility for a Heterogeneous Environment", *Computer*, 21(5), May 1988.

[192] A.P. Black, E.D. Lazowska, H.M. Levy, D. Notkin, J. Sanislo, and J. Zahorjan, "An Approach to Accommodating Hetrogeneity", Technical Report 85–10–04, Dept. of Computer Science, Univ. of Washington, October 1985.

[193] D. Notkin, N. Hutchinson, J. Sanislo, and M. Schwartz, "Accomodating Heterogeneity", *Operating Systems Review*, 20(2):9–22, April 1986.

[194] D. Notkin, A.P. Black, E.D. Lazowska, H.M. Levy, J. Sanislo, and J. Zahorjan, "Interconnecting Heterogeneous Computer Systems", *Communications of the ACM*, 31(3):258–274, 1988.

[195] C.B. Pinkerton et al., "A Heterogeneous Distributed File System", In *Proc. 10th Int. Conf. on Distributed Computing Systems*, pages 424–431, Paris, France, May 1990.

[196] M.F. Schwartz, "Naming in Large Heterogenous Systems", Technical Report 87–08–01, Univ. of Washington, August 1987.

[197] M. Fridrich and W. Older, "The FELIX File Server", In *Proc. 8th Symp. on Operating Systems Principles*, pages 37–44, Pacific Grove, Ca., December 1981.

[198] M. Fridrich and W. Older, "Helix: The Archtecture of the XMS Distributed File System", *IEEE Software*, 2(3):21–29, May 1985.

[199] N. Gammage and L. Casey, "XMS: A Rendezvous-Based Distributed System Software Architecture", *IEEE Software*, 2(3):9–19, May 1985.

[200] D. Comer, "Transparent Integrated Local and Distributed Environment (TILDE) Project Overview", Technical Report CSD-TR-466, Purdue Univ., Dept. of Computer Science, 1984.

[201] J.-F. Pâris and W.F. Tichy, "STORK: An Experimental Migrating File System for Computer Networks", In *Proc. of the IEEE INFOCOM*, IEEE Computer Society Press, pages 168–175, April 1983.

[202] W.F. Tichy, "Towards a Distributed File System", In *Proc. Usenix Summer '84 Conf.*, pages 87–97, Salt Lake City, June 1984.

[203] D.H. Fishman, M.Y. Lai, and W.K. Wilkinson, "An Overview of the JASMIN Database Machine", In *Proc. ACM SIGMOD Conf.*, pages 234–239, Boston, Ma., June 1984.

[204] H. Lee and U. Premkumar, "The Architecture and Implementation of the Distributed JASMIN Kernel", Technical Report TM–ARH–000324, Bell Comm. Research, Morristown, N.J., October 1984.

[205] P. Uppaluru, W.K. Wilkinson, and H. Lee, "Reliable Servers in the JASMIN Distributed System", In *Proc. 7th Int. Conf. on Distributed Computing Systems*, pages 105–111, September 1987.

[206] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel, "LOCUS: A Network Transparent, High Reliability Distributed System", In *Proc. 8th Symp. on Operating Systems Principles*, pages 169–177, ACM SIGOPS, December 1981.

[207] J.D. Moore, "Simple Nested Transactions in LOCUS", Master's thesis, Computer Science Dept., UCLA, Los Angeles, 1982.

[208] A. Goldberg and B. Popek, "Measurement of the Distributed Operating System LOCUS", Technical Report, Univ. of California Los Angeles, 1983.

[209] E. Mueller, J. Moore, and G. Popek, "A Nested Transaction System for LOCUS", In *Proc. 9th Symp. on Operating Systems Principles*, pages 71–89, October 1983.

[210] D.S. Parker, G. Popek, G. Rudisin, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, and C. Kline, "Detection of Mutual Inconsistency in Distributed Systems", *IEEE Transactions on Software Engineering*, SE–9(2):240–247, May 1983.

[211] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS Distributed Operating System", In *Proc. 9th Symp. on Operating Systems Principles*, pages 49–70, October 1983.

[212] B. Walker, *Issues of Network Transparency and File Replication in the Distributed Filesystem Component of LOCUS*, PhD thesis, Computer Science Dept., UCLA, Los Angeles, 1983.

[213] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS Distributed Operating System", *Operating System Review*, 17(5):49–70, 1983.

[214] G.J. Popek and B.J. Walker, *The LOCUS Distributed System Architecture*, MIT Press, Cambridge Massachusetts, 1985.

[215] A. Sheltzer, R. Lindell, and G. Popek, "Name Service Locality and Cache Design in a Distributed Operating System", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 515–523, Cambridge, Massachusetts, May 1986, IEEE.

[216] B. Walker and G. Popek, "The LOCUS Distributed System Architecture", *MIT Press Computer System Series*, page 148, 1985.

[217] R. Wildgruber, "Verteilte UNIX-Systeme", *Markt & Technik Nr. 42*, October 1985.

[218] M. Acetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A new kernel foundation for UNIX development", In *Proc. Usenix Summer '86 Conf.*, pages 93–113, Atlanta, June 1986.

[219] R.V. Baron, R.F. Rashid, E.H Siegel, A. Tevanian, and M.W. Young, "Mach-1: A Multiprocessor Oriented Operating System and Environment", In *New Computing Environments: Parallel, Vector and Systolic*, pages 80–99, SIAM, 1986.

[220] R.V. Baron, D. Black W. Bolosky, J. Chew, D.B. Golub, R.F. Rashid, A. Tevanian, and M.W. Young, "Mach Kernel Interface Manual", Technical Report, Dep. of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA 15213, October 1987.

[221] D.L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System", *IEEE Computer*, 23(5):35–43, May 1990.

[222] R. Draves, M. Jones, and M. Thompson, "MIG — The Mach Interface Generator", Technical Report, Dept. of Comp. Sc., Carnegie-Mellon Univ., Pittsburgh, PA 15213, February 1988.

[223] M.B. Jones, R.F. Rashid, and M.R. Thompson, "Matchmaker: An Interface Specification Language for Distributed Processing", In *Proc. 12th SIGALT/SIGPLAN Symp. on Principles of Programming Languages*, 1985.

[224] M.B. Jones and R.F. Rashid, "Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems", In *Proc. OOPSLA '86 (Objectject-Oriented Programming Systems*, pages 67–77, Portland, Oregon, September 1986.

[225] R.F. Rashid, "From RIG to Accent to Mach: The Evolution of a Network Operating System", In *Proc. Fall Joint Computer Conf.*, pages 1128–1137, Dallas, Texas, November 1986.

[226] R. Rashid, "Mach: Layered Protocols vs. Distributed Systems", In *Proc. COMPCON Spring '87*, pages 82–85, San Francisco, Ca., February 1987.

[227] R.F. Rashid, A. Tevanian, M.W. Young, D.B. Golub, R. Baron, D. Black, W. Bolosky, and J. Chew, "Machine Independant Virtual Memory Management for Paged Uniprocessors and Multiprocessor Architectures", *Operating Systems Review*, 21(4):31–39, October 1987.

[228] R.D. Sansom, D.P. Julin, and R.F. Rashid, "Extending a Capability Based System into a Network Environment", In *Proc. SIGCOMM '86 Symp.*, pages 265–274, Stowe, Vermont, 1986.

[229] A. Tevanian, R.F. Rashid, M.W. Young, D.B. Golub, M.R. Thompson, W. Bolosky, and R. Sanzi, "A UNIX Interface for Shared Memory and Memory Mapped Files Uunder Mach", In *Proc. Usenix Summer '87 Conf.*, pages 53–67, Phoenix, Arizona, June 1987.

[230] A. Tevanian, R.F. Rashid, M.W. Young, D.B. Golub, D.L. Black, and E. Cooper, "Mach Threads and the UNIX kernel: The Battle for Control", In *Proc. Usenix Summer '87 Conf.*, pages 185–197, Phoenix, Arizona, June 1987.

[231] A. Tevanian and R.F. Rashid, "Mach: A Basis for Future UNIX Development", Technical Report, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA 15213, June 1987.

[232] M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black, and R. Baron, "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System", In *Proc. 11th Symp. on Operating Systems Principles*, pages 63–76, 1987.

[233] R.G. Newell, "Solid Modelling and Parametric Design in the Medusa System", *Computer Graphics*, pages 223–236, 1982.

[234] R.G. Newell and G. Parden, "Parametric Design in the Medusa System", *Computer Applications in Production and Engineering*, April 1983.

[235] J.K. Ousterhout, D.A. Scelza, and P.S. Sindhu, "Medusa: An Experiment in Distributed Operating System Structure", *Communications of the ACM*, 23(2):92–105, February 1980.

[236] J.K. Ousterhout, "Medusa: A Distributed Operating System", *Computing Reviews*, 23(5), 1982.

[237] S.R. Ahuja, "S/NET: A High Speed Interconnect for Multiple Computers", *IEEE Journal on Selected Areas in Communications, SAC*, 1(5), November 1983.

[238] N. Carriero and D. Gelernter, "The S/Net's Linda Kernel", *ACM Transactions on Computer Systems*, 4(2):110–129, May 1986.

[239] R.D. Gaglianello and H.P. Katseff, "Meglos: An Operating System for a Multiprocessor Environment", In *Proc. 5th Int. Conf. on Distributed Computing Systems*, pages 35–42, Denver, May 1985.

[240] R.D. Gaglianello and H.P. Katseff, "The Meglos User Interface", In *Proc. Fall Joint Computer Conf.*, pages 169–177, Dallas, Texas, November 1986.

[241] R.D. Gaglianello and H.P. Katseff, "A Distributed Computing Environment for Robotics", In *Proc. 1986 Int. Conf. on Robotics and Automation*, pages 1890–1896, San Francisco, April 1986.

[242] A. Barak and Z. Drezner, "Distributed Algorithms for the Average Load of a Multicomputer", Technical Report CRL–TR–17–84, Computing Research Laboratory, The Univ. of Michigan, March 1984.

[243] A. Barak and A. Shiloh, "A Distributed Load-Balancing Policy for a Multicomputer", *Software — Practice and Experience*, 15:901–913, September 1985.

[244] A. Barak and A. Littmann, "A Multicomputer Distributed Operating System", *Software — Practice and Experience*, 15(8):727–737, August 1985.

[245] A. Barak and O.G. Paradise, "MOS – A Load-Balancing UNIX", *Proc. EUUG Autumn '86*, pages 273–280, September 1986.

[246] A. Barak and D. Malki, "Distributed Light Weighted Processes in MOS", In *Proc. EUUG Autumn '88*, pages 335–343, Cascais, Portugal, October 1988.

[247] Z. Drezner and A. Barak, "A Probabilistic Algorithm for Scattering Information in a Multicomputer System", Technical Report CRL–TR–15–84, Computing Research Laboratory, The Univ. of Michigan, March 1984.

[248] R. Hofner, "The MOS Communication System", Master's thesis, Dept. of Computer Science, The Hebrew Univ. of Jerusalem, February 1984.

[249] J. Masel, "The MOS Filing System", Master's thesis, Dept. of Computer Science, The Hebrew Univ. of Jerusalem, August 1983.

[250] A. Shiloh, "Load Sharing in a Distributed Operating System", Master's thesis, Dept. of Computer Science, The Hebrew Univ. of Jerusalem, July 1983.

[251] T.H. Dineen, P.J. Leach, N.W. Mishkin, J.N. Pato, and G.L. Wyant, "The Network Computing Architecture and System: An Environment for Developing Distributed Applications", In *Proc. Usenix Summer '87 Conf.*, pages 385–398, Phoenix, Arizona, June 1987.

[252] T.H. Dineen, P.J. Leach, N.W. Mishkin, J.N. Pato, and G.L. Wyant, "The Network Computing Architecture and System: An Environment for Developing Distributed Applications", In *Proc. COMPCON Spring '88*, pages 296–299, San Francisco, Ca., March 1988.

[253] W. Bronsvoort, "GUTS, a Multi-User Operating System for the PDP11/40", Technical Report TW–200, Rijksuniversiteit te Groningen, Mathematisch Institut, 1978.

[254] D.R. Brownbridge, L.F. Marshall, and B. Randell, "The Newcastle Connection or UNIXes of the World Unite", *Software — Practice and Experience*, 12:1147–1162, 1982.

[255] A. Linton and F. Panzieri, "A Communication System Supporting Large Datagrams on a Local Area Network", Technical Report SRM/193, Univ. of Newcastle upon Tyne, May 1984.

[256] F. Panzieri, "Software and Hardware of High Integrity Systems Projects: Final Report", Technical Report ERC/9/4/2043/059 RSRE, Univ. of Newcastle upon Tyne, June 1982.

[257] F. Panzieri and B. Randell, "Interfacing UNIX to Data Communications Networks", Technical Report 190, Univ. of Newcastle upon Tyne, December 1983.

[258] J. Rushby and B. Randell, "A Distributed Secure System", *IEEE Computer*, pages 55–67, July 1983.

[259] C.R. Snow and H. Whitfield, "An Experiment with the Newcastle Connection Protocol", *Software — Practice and Experience*, 16(11):1031–1043, November 1986.

[260] M.V. Wilkes and D.J. Wheeler, "The Cambridge Digital Communication Ring", In *Proc. Local Area Communications Network Symp.*, U. S. National Bureau of Standards Special Publication, Boston, 1979.

[261] A. Tripathi, A. Ghonami, and T. Schmitz, "Object Management in the NEXUS Distributed Operating System", In *Proc. IEEE COMPCON '87*, pages 50–53, February 1987.

[262] A. Tripathi, "NEXUS Distributed Operating System", In *Tutorial No. 4, Distributed Operating Systems, 7th Int. Conf. Distributed Computing Systems*, pages 142–159, Berlin, West Germany, September 1987.

[263] B. Fraser-Campbell and M.B. Rosen, "An Implementation of NFS under System V.2", *Proc. EUUG*, April 1986.

[264] M.J. Hatch, M. Katz, and J. Rees, "AT&T's RFS and SUN's NFS — A Comparison of Heterogeneous Distributed File Systems", *UNIX WORLD*, pages 159–164, December 1985.

[265] B. Lyon, "Sun Remote Procedure Call Specification", Technical Report, Sun Microsystems, Inc., 1984.

[266] B. Lyon, "Sun External Data Representation Specification", Technical Report, Sun Microsystems, Inc., 1984.

[267] B. Lyon, G. Sager, J.M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, D. Walsh, and P. Weiss, "Overview of the SUN Network File System", In *Proc. Usenix Conf.*, pages 1–8, Dallas, January 1985.

[268] R. Sandberg, "The SUN Network File System: Design, Implementation and Experience", *SUN Microsystems, Inc.*, 1985.

[269] R. Sandberg, "Sun Network Filesystem Protocol Specification", Technical Report, Sun Microsystems, Inc., 1985.

[270] S. Spanier, "Comparing Distributed File Systems", *Data Communications*, pages 173–186, December 1987.

[271] SUN Microsystems, Inc., *Remote Procedure Call Reference Manual*, October 1984.

[272] SUN Microsystems, Inc., *External Data Representation Reference Manual*, October 1984.

[273] SUN Microsystems, Inc., *Remote Procedure Call Protocol Specification*, October 1984.

[274] P. Weiss, "Yellow Pages Protocol Specification", Technical Report, Sun Microsystems, Inc., 1985.

[275] A. West, "The SUN Network File System, NFS – Business Overview", *SUN Microsystems Inc.*, 1985.

[276] J. Kaiser, E. Nett, and R. Kroeger, "Mutabor — A Coprocessor Supporting Object-Oriented Memory-Management and Error Recovery", *HICSS-21*, January 1988.

[277] M. Mock, "Globales Objektmanagement in einem verteilten onjektorientierten System", GMD-Studien 128, Gesellschaft für Mathematik und Datenverarbeitung mbH, GMD, St. Augustin, West Germany, December 1987.

[278] E. Nett, K.E. Grosspietsch, A. Jungblut, J. Kaiser, R. Kroeger, W. Lux, M. Speicher, and H.W. Winnebeck, "PROFEMO: Design and Implementation of a Fault Tolerant Distributed System Architecture", Technical Report GMD-Studien Nr. 100, Gesellschaft für Mathematik und Datenverarbeitung mbH, GMD, St. Augustin, West Germany, June 1985.

[279] E. Nett, K.E. Grosspietsch, A. Jungblut, J. Kaiser, R. Kroeger, W. Lux, M. Speicher, and H.W. Winnebeck, "PROFEMO: Design and Implementation of a Fault Tolerant Distributed System Architecture (Formal Specification)", Technical Report GMD-Studien Nr. 101, Gesellschaft für Mathematik und Datenverarbeitung mbH, GMD, St. Augustin, West Germany, June 1985.

[280] E. Nett, J. Kaiser, and R. Kroeger, "Providing Recoverability in a Transaction Oriented Distributed Operating System", In *Proc 6th Int. Conf. on Distributed Computing Systems*, May 1986.

[281] R. Schumann, "Transaktions-Verwaltung in einem verteilten objektorientierten System", GMD-Studien 134, Gesellschaft für Mathematik und Datenverarbeitung mbH, GMD, St. Augustin, West Germany, January 1988.

[282] D. Keefe, G.M. Tomlinson, I.C. Wand, and A.J. Wellings, *PULSE: An Ada-Based Distributed Operating System*, Academic Press, London, 1985.

[283] A.J. Wellings, *Distributed Operating Systems and the Ada Programming Language*, PhD thesis, Dept. of Computer Science, Univ. of York, April 1984.

[284] A.J. Wellings, "Communication between Ada Programs", In *Ada Applications and Environments*, pages 143–153, St. Paul, Minnesota, October 1984.

[285] A.J. Wellings, "The PULSE Project", In *ACM SIGOPS Workshop on Operating Systems in Computer Networks*, Rüschlikon, Switzerland, January 1985, Operating Systems Review, 19(2):6–40, April 1985.

[286] J.F. Cabrera and J. Wyllie, "QuickSilver Distributed File Services: An Architecture for Horizontal Growth", Technical Report RJ 5578 (56697), IBM Almaden Research Center, San Jose, January 1987.

[287] R. Haskin, Y. Malachi, W. Sawdon, and G. Chan, "Recovery Management in QuickSilver", *ACM Transactions on Computer Systems*, 6(1):82–108, February 1988.

[288] M.J. Bach, M.W. Luppi, A.S. Melamed, and K. Yueh, "A Remote File Cache for RFS", In *Proc. Usenix Summer '87 Conf.*, pages 273–280, Phoenix, Arizona, June 1987.

[289] R. Hamilton, P. Krzyzanowski, M. Padovano, and M. Purdome, "An Administrator's View of RFS", *AT & T Information Systems*, June 1986.

[290] D.P. Kingston, "Remote File Systems on UNIX", *Proc. EUUG Copenhagen*, pages 77–93, September 1985.

[291] A.P. Rifkin, M.P. Forbes, R.L. Hamilton, M. Sabrio, S. Shah, and K. Yueh, "RFS Architectural Overview", *Proc. EUUGN*, 6(2):13–23, 1986.

[292] D.M. Ritchie, "A Stream Input-Output System", *AT&T Bell Labs Technical Journal*, 63(8), October 1984.

[293] G.R. Andrew, "Distributed Systems Research at Arizona", In *2nd ACM SIGOPS European Workshop on "Making Distributed Systems Work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[294] G.R. Andrew, R.D. Schlichting, R. Hayes, and T. Purdin, "The Design of the Saguaro Distributed Operating System", *IEEE Transactions on Software Engineering*, SE–13(1):104–118, January 1987.

[295] R.D. Schlichting, G.R. Andrews, and T. Purdin, "Mechanisms to Enhance File Availability in Distributed Systems", In *Proc. 16th Int. Symp. Fault-Tolerant Computing*, pages 44–49, Vienna, July 1986.

[296] G.L. Chesson, "Datakit Software Architecture", In *Proc. ICC*, pages 20.2.1–20.2.5, Boston, Ma., June 1979.

[297] A.G. Fraser, "Datakit – A Modular Network for Synchronous and Asynchromous Traffic", In *Proc ICC*, pages 20.1.1–20.1.3, Boston, Ma., June 1979.

[298] G.W.R. Luderer, H. Che, J.P. Haggerty, P.A. Kirslis, and W.T. Marshall, "A Distributed UNIX System Based on a Virtual Circuit Switch", In *Proc. 8th Symp. on Operating Systems Principles*, pages 160–168, December 1981.

[299] G.W.R. Luderer, H. Che, and W.T. Marshall, "A Virtual Circuit Switch as the Basis for a Distributed System", *7th Data Communications Symp., ACM, IEEE Computer Society, IEEE Communications Society*, pages 27–29, October 1981.

[300] M. Hurwicz, "MS-DOS 3.1 Makes it Easy to Use IBM PCs on a Network", *Data Communications*, November 1985.

[301] IBM, "IBM Personal Computer Seminar Proceedings", Technical Report, May 1985.

[302] M. Stieglitz, "IBM Provides Industry with a Versatile Local Network Standard", *Data Communications*, June 1985.

[303] M. Makpangou and M. Shapiro, "The SOS Object-Oriented Communication Service", In *Proc. Int. Conf. on Computers and Communications*, Tel Aviv, Israel, October 1988.

[304] M. Shapiro, "SOS: A Distributed Object-Oriented Operating System", In *2nd ACM SIGOPS European Workshop on "Making Distributed Systems Work"*, Amsterdam, Netherlands, September 1986, Operating Systems Review, 21(1):49–84, January 1987.

[305] M. Shapiro, "Structure and Encapsulation in Distribute Systems: the Proxy Principle", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 198–204, Cambridge, Mass., May 1986, IEEE.

[306] M. Shapiro, V. Abrossimov, P. Gautron, S. Habert, and M. Makpangou, "Un Recueil de Papiers sur le Système d'Exploitation Réparti à Objets SOS", Rapport Technique 84, Institut National de la Recherche en Informatique et Automatique, Rocquencourt (France), May 1987.

[307] M. Shapiro and S. Habert, *Programmer's Manual for SOS Prototype Version 2/3*, 1988, to appear.

[308] F. Douglis and J. Ousterhout, "Process Migration in the Sprite Operating System", In *Proc. 7th Int. Conf. on Distributed Computing Systems*, pages 18–25, Berlin, West Germany, September 1987.

[309] M.N. Nelson, "Virtual Memory for the Sprite Operating System", Technical Report UCB/CSD 86/301, Computer Science Division, Univ. of California Berkeley, Berkeley, California 94720, June 1986.

[310] M. Nelson, B. Welch, and J.K. Ousterhout, "Caching in the Sprite Network File System", *ACM Transactions on Computer Systems*, 6(1):134–154, February 1987.

[311] J. Ousterhout, A. Cherenson, F. Douglas, M. Nelson, and B. Welch, "The Sprite Network Operating System", Technical Report UCB/CSD 87/359, Computer Science Division, Univ. of California at Berkeley, Berkeley, Ca., 1987.

[312] J. Ousterhout, "Position Statement for Sprite: The File System as the Center of a Network Operating System", In *Proc. Workshop on Workstation Operating Systems*, Cambridge, MA, November 1987, IEEE Computer Society Technical Commitee on Operating Systems.

[313] B.B. Welch, "The Sprite Remote Procedure Call System", Technical Report UCB/CSD 86/302, Computer Science Division, Univ. of California Berkeley, Berkeley, California 94720, June 1986.

[314] B. Welch and J. Ousterhout, "Prefix Tables: A Simple Mechanism for Locating Files in a Distributed Filesystem", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 184–189, New York, May 1986, IEEE Computer Society Press.

[315] G.C. Arens, "Recovery of the SWALLOW Repository", Technical Report MIT/LCS/TR–252, Lab. for Computer Science, MIT, Cambridge, Mass., January 1981.

[316] D.P. Reed and L. Svobodova, "SWALLOW: A Distributed Data Storage System for a Local Network", In A. West and P. Janson, editors, *Local Networks for Computer Communications*, pages 355–373, North-Holland Publishing Company, 1981.

[317] L. Svobodova, "Management of the Object Histories in the SWALLOW Repository", Technical Report MIT/LCS/TR–243, Lab. for Computer Science, MIT, Cambridge, Mass., July 1980.

[318] D.R. Cheriton, "Local Networking and Internetworking in the V-System", In *Proc. 7th Data Communications Symp.*, pages 9–16, ACM/IEEE, October 1983.

[319] D.R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations", In *Proc. 9th Symp. on Operating Systems Principles*, pages 129–140, ACM, October 1983.

[320] D.R. Cheriton, "An Experiment using Registers for Fast Message-based Interprocess Communication", *Operating Systems Review*, 18(4):12–20, October 1984.

[321] D.R. Cheriton, "The V Kernel: A Software Base for Distributed Systems", *IEEE Software*, 1(2):19–42, April 1984.

[322] D.R. Cheriton, "Uniform Access to Distributed Name Interpretation in the V-System", In *Proc. 4th Int. Conf. on Distributed Computing Systems*, pages 290–297, IEEE, May 1984.

[323] D.R. Cheriton and W. Zwaenepoel, "One-to-many Interprocess Communication in the V-System", In *SIGCOMM '84 Symp. in Communications Architectures and Protocols*, ACM, June 1984.

[324] D.R. Cheriton, "Request-Response and Multicast Interprocess Communication in the V Kernel", In *Networking in Open Systems*, pages 296–312, LNCS #248, Springer Verlag, 1986.

[325] D.R. Cheriton, "Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems Design", In *Proc. 6th Int. Conf. on Distributed Computing Systems*, pages 190–197, IEEE, May 1986.

[326] D.R. Cheriton, "VMTP: A Transport Protocol for the Next Generation of Communication Systems", In *Proc. SIGCOMM '86*, Stowe, Vt., August 1986, ACM.

[327] D.R. Cheriton, "UIO: A Uniform I/O System Interface for Distributed Systems", *ACM Transactions on Computer Systems*, 5(1):12–46, February 1987.

[328] D.R Cheriton and C. Williamson, "Network Measurement of the VMTP Request-Response Protocol in the V Distributed System", In *Proc. SIGMETRICS 87*, Banff, Canada, 1987, ACM.

[329] D. Cheriton, "The V Distributed System", *Communications of the ACM*, 31(3):314–333, March 1988.

[330] D.R. Cheriton, "Unified Management of Memory and File Caching Using the V Virtual Memory System", Technical Report STAN–CS–88–1192, Dept. of Computer Science, Stanford Univ., 1988.

[331] D.R. Cheriton and T.P. Mann, "Decentralizing: A Global Name Service for Efficient Fault-tolerant Access", *ACM Transactions on Computer Systems*, 1988, to appear.

[332] K.A. Lantz and W.I. Nowicki, "Structured Graphics for Distributed Systems", *ACM Transactions on Computer Systems*, 3(1):23–51, January 1984.

[333] K.A. Lantz and W.I. Nowicki, "Virtual Terminal Services in Workstation-Based Distributed Systems", In *Proc. 17th Hawaii Int. Conf. on System Sciences*, pages 196–205, ACM/IEEE, January 1984.

[334] M. Stumm, "Verteilte Systeme: Eine Einführung am Beispiel V", *Informatik Spektrum*, 10(5):246–261, October 1987.

[335] M.M. Theimer, K.A. Lantz, and D.R. Cheriton, "Preemptable Remote Execution Facility in the V-System", In *Proc. 10th Symp. on Operating Systems Principles*, ACM SIGOPS, 1985.

[336] W. Zwaenepoel, "Implementation and Performance of Pipes in the V-System", *IEEE Transactions on Computers*, c–34(12):1174–1178, December 1985.

[337] N.P. Kronenberg, H.M. Levy, and W.D. Strecker, "VAXclusters: A Closely–Coupled Distributed System", *ACM Transactions on Computer Systems*, 4(2):130–146, May 1986.

[338] C. Dellar, "A File Server for a Network of Low Cost Personal Microcomputers", *Software — Practice and Experience*, 12:1051–1068, December 1982.

[339] R.A. Finkel and M.H. Solomon, "The Arachne Kernel, Version 1.2", Technical Report 380, Univ. of Wisconsin–Madison Computer Sciences, April 1980.

[340] R.A. Finkel and M.H. Solomon, "The Arachne Distributed Operating System", Technical Report 439, Univ. of Wisconsin–Madison Computer Sciences, July 1981.

[341] S. Muir, D. Hutchison, and D. Shepherd, "Arca: A Local Network File Server", *The Computer Journal*, 28(3):243–249, March 1985.

[342] E.D. Jensen, "ArchOS", In *ACM SIGOPS Workshop on Operating Systems in Computer Networks*, Rüschlikon, Switzerland, January 1985, Operating Systems Review, 19(2):6–40, April 1985.

[343] A.Z. Spector, "Camelot: A Flexible, Distributed Transaction Processing System", In *Proc. Spring Compcon 88, 33rd IEEE Computer Society Int. Conf.*, pages 432–436, San Fransisco, Ca., March 1988.

[344] M.B. Yelavich, "Customer Information Control System — An Evolving System Facility", *IBM Systems Journal*, 24(4):263–278, 1985.

[345] E.C. Cooper, "Replicated Distributed Programs", Technical Report UCB/CSD 85/231, Computer Science Division Univ. of Berkeley, Ca, May 1985.

[346] E. Cooper, "Replicated Distributed Programs", *Operating Systems Review*, 19(5):63–78, December 1985.

[347] D.C. Oppen and Y.K. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment", *ACM Transactions on Office Information Systems*, 1(3):230–253, July 1983.

[348] E.J. Ball, M.R. Barbacci, S.E. Fahlman, S.P. Harbison, P.G. Hibbard, R.F. Rashid, G.G. Robertson, and G.L. Steele Jr., "The Spice Project", *Computer Science Research Review 1980–1981*, pages 5–36, 1982, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Penn.

[349] L.A. Rowe and K.P. Birman, "A Local Network Based on the UNIX Operating System", *IEEE Transactions on Software Engineering*, SE-8(2):137–146, March 1982.

[350] M. Satyanarayanan, J.J. Kistler, and E.H. Siegel, "Coda: A Resilient Distributed File System", In *Proc. Workshop on Workstation Operating Systems*, Cambridge, MA, November 1987, IEEE.

[351] J. Magee, J. Kramer, and M. Sloman, "The CONIC Support Environment for Distributed Systems", In Y. Paker et al., editor, *Distributed Operating Systems: Theory and Practice*, volume F28 of *NATO ASI series*, pages 289–311, Springer Verlag, August 1986.

[352] D. Anderson, D. Ferrari, P.V. Rangan, and S.-Y. Tzou, "The DASH Project: Issues in the Design of a Very Large Distributed System", Technical Report UCB/CSD 87/338, Univ. of California, Berkeley, 1987.

[353] D.P. Anderson et al., "Support for Continuous Media in the Dash System", In *Proc. 10th Int. Conf. on Distributed Computing Systems*, pages 54–61, Paris, France, May 1990.

[354] T. Marill and D. Stern, "The Datacomputer — A Network Data Utility", In *Proc. AFIPS NCC 44*, pages 389–395, 1975.

[355] F. Baskett, J.H. Howard, and J.T. Montague, "Task Communications in DEMOS", In *Proc. 6th Symp. on Operating Systems Principles*, pages 23–32, Purdue Univ., November 1977.

[356] M.J. Arden, "DFS925: A Distributed File System in a Workstation/LAN Environment", Technical Report UCB/CSD 85/236, Computer Science Division, Univ. of Berkeley, Ca., May 1985.

[357] J. Nehmer, "DISTOS", In *ACM SIGOPS Workshop on Operating Systems in Computer Networks*, Rüschlikon, Switzerland, January 1985, Operating Systems Review, 19(2):6–40, April 1985.

[358] D. Christie, "DISTRIX", In *ACM SIGOPS Workshop on Operating Systems in Computer Networks*, Rüschlikon, Switzerland, January 1985, Operating Systems Review, 19(2):6–40, April 1985.

[359] D.C. Daniels, "Dragon Slayer: A Blueprint for the Design of a Completely Distributed Operating System", Master's thesis, Wayne State University, Detroit, 1986.

[360] J. Alberi and M. Pucci, "The DUNE Distributed Operating System", Technical Report TM-ARH-010641 (technical memorandum), Bellcore, December 1987.

[361] J.-P. Banâtre, M. Banâtre, G. Lapalme, and F. Ployette, "The Design and Building of Enchère, a Distributed Electronic Marketing System", *Communications of the ACM*, 29(1):19–29, January 1986.

[362] A. Borr, "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing", In *Proc. 7th Int. Conf. on Very Large Data Bases*, Cannes, France, September 1981, also TANDEM TR 81.2, Tandem Computers Inc., Cupertino, Ca., June 1981.

[363] C.P. Thacker and L.C. Stewart, "Firefly: A Multiprocessor Workstation", In *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 164–172, Palo Alto, Ca., February 1987.

[364] X. Jia et al., "Highly Concurrent Directory Management in the Galaxy Distributed System", In *Proc. 10th Int. Conf. on Distributed Computing Systems*, pages 416–423, Paris, France, May 1990.

[365] R. Rodriguez, M. Koehler, and R. Hyde, "The Generic File System", In *Proc. EUUG Autumn '86*, pages 83–92, Manchester, UK, September 1986.

[366] J.H. Carson, "A Distributed Operating System for a Workstation Environment", In *Proc. Phoenix Conf. Computers and Communication*, pages 213–217, March 1988.

[367] C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sprol, and D.R. Boogs, "Alto: A Personal Computer", In D.P. Siewiorek, C.G. Bell, and A. Newell, editors, *Computer Structures: Principles and Examples*, McGraw-Hill, New York, N.Y., 2nd edition, 1981.

[368] K. Birman, "Replication and Fault-Tolerance in the ISIS System", *Operating Systems Review*, 19(5):79–86, December 1985.

[369] A. Langsford, "MANDIS: An Experiment in Distributed Processing", In R. Speth, editor, *Research into Networks and Distributed Applications*, pages 787–794, North-Holland, 1988.

[370] L.D. Wittie and A.M. van Tilborg, "MICROS, a Distributed Operating System for MICRONET, a Reconfigurable Network Computer", *IEEE Transactions on Computers*, c-29(12):1133–1144, December 1980.

[371] R. Agrawal and A.K. Ezzat, "Location Independent Remote Execution in NEST", *IEEE Transactions on Software Engineering*, SE–13(8):905–912, August 1987.

[372] J.F. Bartlett, "A NonStop Kernel", In *Proc. 8th Symp. on Operating Systems Principles*, pages 22–29, Pacific Grove, Ca., December 1981.

[373] B.G. Lindsay, L.M. Haas, C. Mohan, P.F. Wilms, and R.A. Yost, "Computation and Communication in $R\star$: A Distributed Database Manager", *ACM Transactions on Computer Systems*, 2(1):24–38, February 1984.

[374] R. Finkel and M. Solomon, "The Roscoe Kernel", Technical Report 337, Computer Sciences Dept., Univ. of Wisconsin-Madison, October 1978.

[375] M. Solomon and R. Finkel, "Roscoe: A Multi-Microcomputer Operating System", Technical Report 321, Computer Sciences Dept., Univ. of Wisconsin-Madison, April 1978.

[376] R. Tischler, M. Solomon, and R. Finkel, "Roscoe Users Guide", Technical Report 336, Computer Science Dept., Univ. of Wisconsin-Madison, October 1978.

[377] R. Tischler, R. Finkel, and M. Solomon, "Roscoe Utility Processes", Technical Report 338, Computer Science Dept., Univ. of Wisconsin-Madison, October 1978.

[378] J.N. Gray, P. McJones, M.W. Blasgen, R.A. Lorie, T.G. Price, G.F. Putzulu, and I.L. Traiger, "The Recovery Manager of the System R Database Manager", *ACM Computing Surveys*, 13(2):223–242, June 1981.

[379] C.H. Sauer, D.W. Johnson, L.K. Loucks, A.A. Shaheen-Gouda, and T.A. Smith, "RT PC Distributed Services Overview", *Operating Systems Review*, 21(3):18–29, July 1987.

[380] M.R. Thompson, R.D. Sansom, M.B. Jones, and R.F. Rashid, "Sesame: The Spice File System", Technical Report CMU–CS–85–172, Computer Science Dept., Carnegie-Mellon Univ., December 1985.

[381] H.M. Levy, "The StarOS System", In *Capability-based Computer Systems*, pages 127–137, Digital Press, 1984.

[382] D.R. Cheriton, M.A. Malcolm, L.S. Melen, and G.R. Sager, "Thoth, a Portable Real-Time Operating System", *Communications of the ACM*, 22(2):105–115, February 1979.

[383] P. McJones and A. Hisgen, "The Topaz System: Distributed Multiprocessor Personal Computing", In *Proc. Workshop on Workstation Operating Systems*, Cambridge, MA, November 1987.

[384] R.P. Hughes, "The Transparent Remote File System", In *Proc. Usenix Summer '86 Conf.*, pages 306–317, Atlanta, June 1986.

[385] G.D. Burns, "Trollius: Early American Transputer Software", The Ohio State University, Columbus, Ohio, 1988.

[386] W.G. Giloi and P.M. Behr, "Obtaining a Secure, Fault-Tolerant Distributed System with Maximal Performance", In *Proc. IFIP Workshop on Hardware Supported Implementation of Concurrent Languages in Distributed Systems*, pages 101–114, Bristol, 1984.

[387] D. Swinehart, G. McDaniel, and D. Boggs, "WFS: A Simple Shared File System for a Distributed Environment", In *Proc. 7th Symp. on Operating System Principles*, pages 9–17, December 1979.

[388] F. Tisato and R. Zicari, "The Xcode Machine", In *Proc. 3rd Symp. on Microcomputer and Microprocessors Applications*, Budapest, Hungary, October 1983, also in ACM SIGSMALL Newsletter, 10, 1, 1984.

[389] W. Bux, F. Closs, P. Janson, K. Kuemmerle, H.R. Mueller, and E.H. Rothhauser, "A Local-Area Communication Network Based on a Reliable Token-Ring System", In *Proc. Int. Symp. on Local Computer Networks*, pages 69–82, Florence, Italy, April 1982.