Unit 15:
Experimental Microkernel Systems

15.1. The Amoeba Distributed Operating System

The Amoeba Distributed Operating System

- Research vehicle for distributed and parallel operating systems, runtime systems, languages, and applications.
- Developed at Vrije Universiteit, Amsterdam
  - (A.S. Tanenbaum, F. Kaashoek, S.J. Mullender, R.v. Renesse)
  - www.cs.vu.nl/pub/amoeba/amoeba.html

80 processor Amoeba Sparc Cluster at Vrije Universiteit

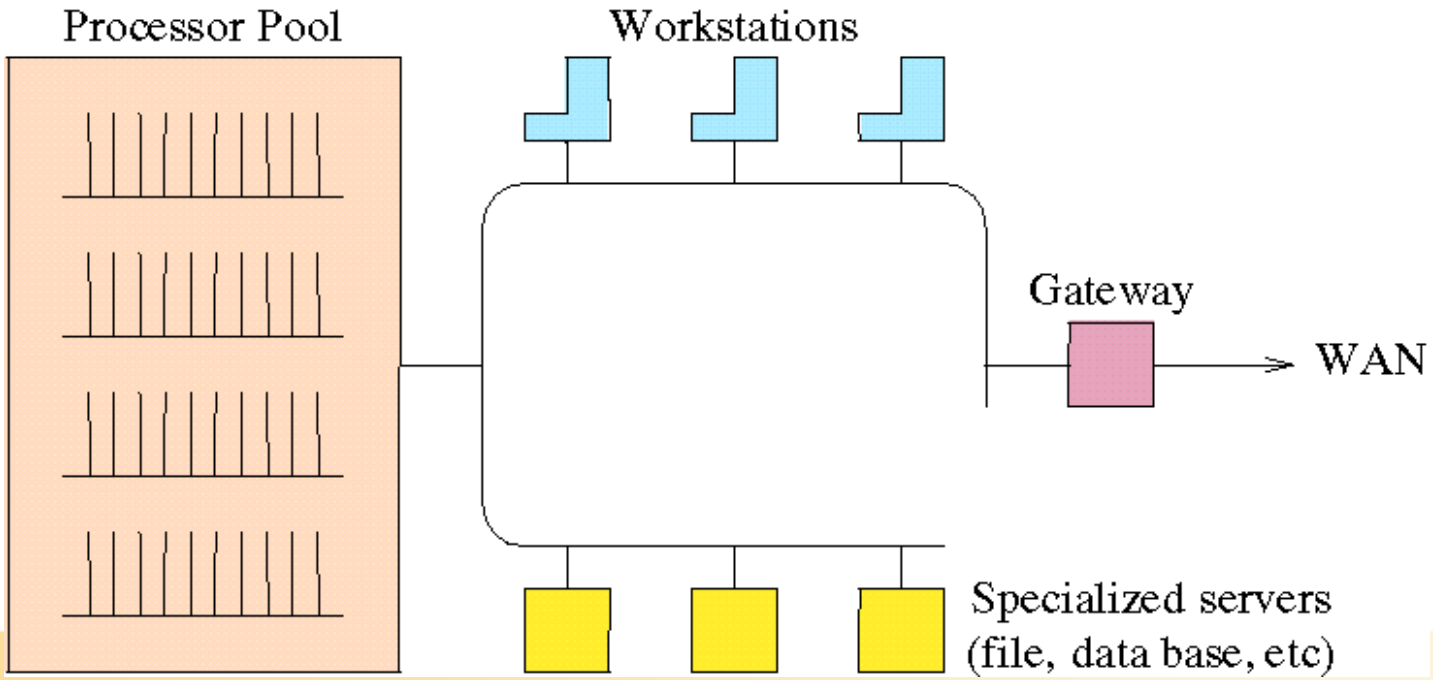
Research Goalds

- Transparent distributed operating system
- UNIX look and feel to the user
- Actions make use of multiple machines distributed over the network
  - Process servers
  - File servers
  - Directory servers
  - Compute servers
- No home machine; machines do not have owners
- Load balancing, processor pools

Amoeba System Architecture

Two basic assumptions:

1. Systems will have a very large number of CPUs
2. Each CPU will have tens of Mbyts of memory



Amoeba Architecture (1)

- The Amoeba architecture consists of four principal components
- First are the workstations, one per user, on which users can carry out editing and other tasks that require fast interactive response.
- Workstations are all diskless, and are primarily used as intelligent terminals that do window management.
  - Sun3's and VAXstations have been used as workstations, as well as
  - X-terminals.

Amoeba Architecture (2)

- Second are the pool processors, a group of CPUs that can be dynamically allocated as needed, used, and then returned to the pool.
  - The make command might need to do six compilations, so six processors could be taken out of the pool for the time necessary to do the compilation and then returned.
  - Alternatively, with a five-pass compiler, 5 x 6 = 30 processors could be allocated for the six compilations, gaining even more speedup.
- Many applications, use large numbers of pool processors to do their computing
  - heuristic search in AI applications (e.g., playing chess),
- Pool processors can be heterogeneous

Amoeba Architecture (3)

- Third are the specialized servers,
  - such as directory servers,
  - file servers,
  - data base servers,
  - boot servers, and various other servers with specialized functions.
- Each server is dedicated to a specific function.
  - In some cases, there are multiple servers that provide the same function,
  - The replicated file system is an example.

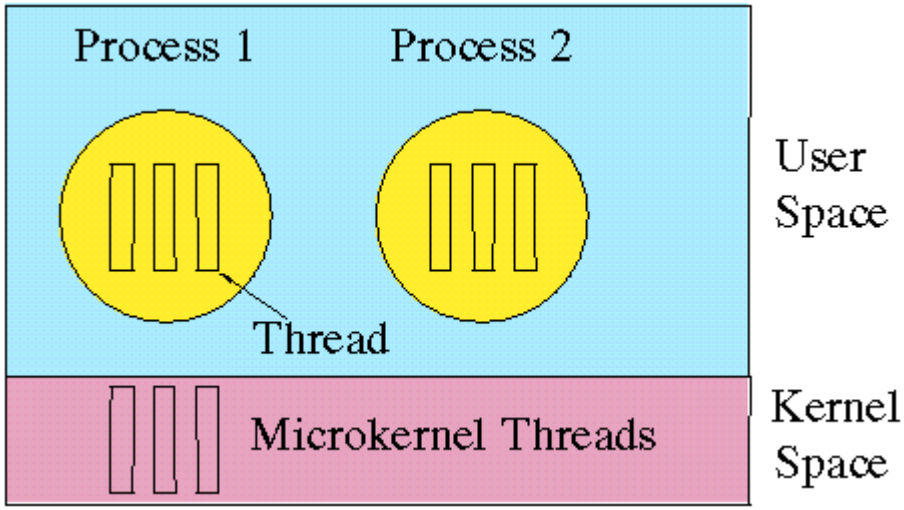
Amoeba Architecture (4)

- Gateways link Amoeba systems at different sites into a single, uniform system.
- Gateways isolate Amoeba from WAN-protocols
- All the Amoeba machines run the same kernel
  - multithreaded processes,
  - communication services,
  - I/O, and little else.
- The basic idea behind the kernel was to keep it small
  - enhance its reliability,
- As much OS functionality as possible is provided in user-space
  - flexibility and ease of experimentation.

The Amoeba Microkernel

The Amoeba microkernel runs on all machines in the system. It has four primary functions:

1. Manage processes and threads within these processes.
2. Provide low-level memory management support.
3. Support transparent communication between arbitrary threads.
4. Handle I/O.

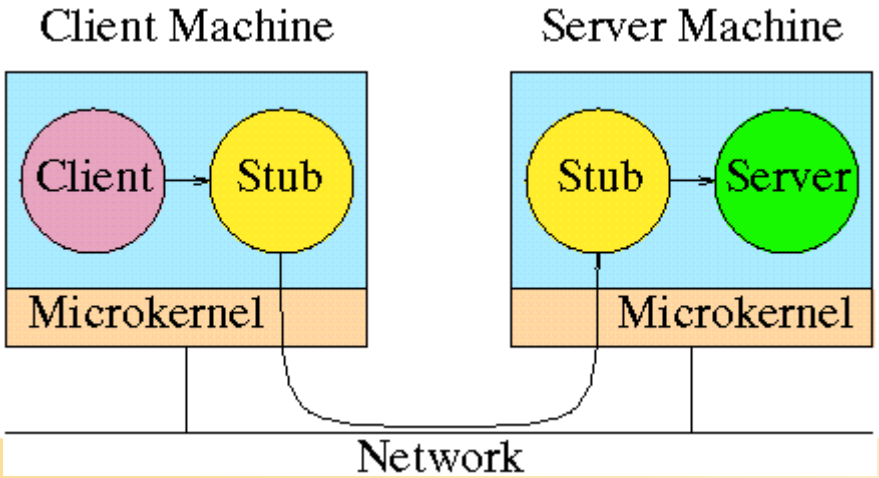


Threads in Amoeba

- Like most OS, Amoeba supports the process concept.
- Amoeba also supports multiple threads within a single address space.
  - A process with one thread is essentially the same as a UNIX process.
  - Such a process has a single address space, a set of registers, a program counter, and a stack.
- Threads are managed and scheduled by the microkernel
  - The primary argument for making the threads known to the kernel rather than being pure user concepts relates to our desire to have communication be synchronous (i.e., blocking).

RPC Communication

- All RPCs are from one thread to another.
  - User_to_user, user_to_kernel, and kernel_to_kernel communication all occur.
  - (Kernel_to_user is technically legal, but, since that constitutes an upcall, they have been avoided except where that was not feasible).
  - When a thread blocks awaiting the reply, other threads in the same process that are not logically blocked may be scheduled.



RPC Addressing

- Addressing is done by allowing any thread to choose a random 48bit number called a port .
  - All messages are addressed from a sending port to a destination port.
  - A port is nothing more than a kind of logical thread address.
  - There is no data structure and no storage associated with a port.
- When an RPC is executed, the sending kernel locates the destination port by broadcasting a special LOCATE message, to which the destination kernel responds.
  - Once this cycle has been completed, the sender caches the port, to avoid subsequent broadcasts.
- The RPC mechanism makes use of three principal kernel primitives:
  - Do_remote_op() - send a message from client to server and wait for the reply
  - Get_request() - indicates a server's willingness to listen on a port
  - Put_reply() - done by a server when it has a reply to send

RPC Performance

- All communication in Amoeba is based on RPC.
  - If the RPC is slow, everything built on it will be slow too (e.g., the file server performance).
  - For this reason, considerable effort has been spent to optimize the performance of the RPC between a client and server running as user processes on different machines, as this is the normal case in a distributed system.

	Delay (msec)			*Bandwidth (Kbytes/sec)*		
	case 1 *(4 bytes)*	case 2 *(8 Kb)*	case 3 *(30 Kb)*	case 1 *(4 bytes)*	case 2 *(8 Kb)*	case 3 *(30 Kb)*
pure Amoeba local	0.5	2.0	5.6	7.4	4000	5232
pure Amoeba remote	1.1	11.1	37.4	3.5	721	783
UNIX driver local	4.2	7.7	17.0	0.9	1039	1723
UNIX driver remote	5.1	26.7	88.6	0.8	300	331
Sun RPC local	5.7	12.8	imposs.	0.7	625	imposs.
Sun RPC remote	6.7	24.6	imposs.	0.6	325	imposs.
	(a)			(b)		

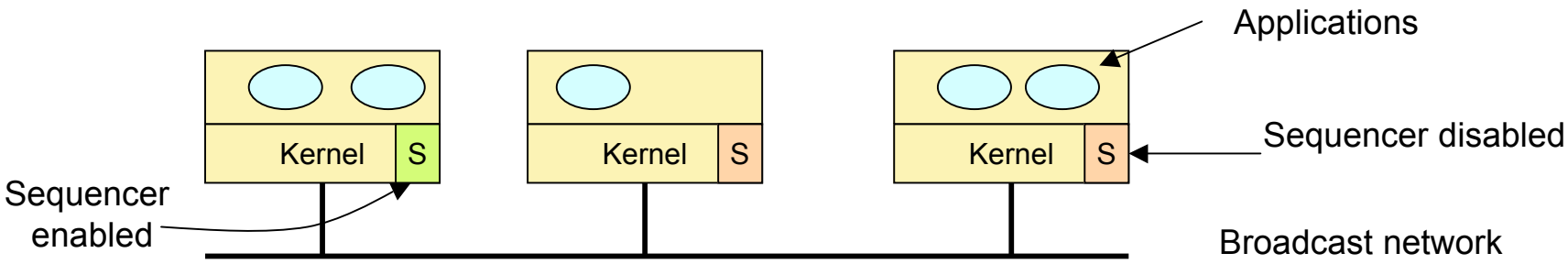
Group Communication in Amoeba

Call	Description
CreateGroup	Create a new group and set its parameters
JoinGroup	Make the caller a member of a group
LeaveGroup	Remove the caller from a group
SendToGroup	Reliably send a message to all members of a group
ReceiveFromGroup	Block until message arrives from a group
ResetGroup	Initiate recovery after a process crash

- A group in Amoeba consists of one or more processes that are cooperating to provide a service
- Processes can be members of several groups
- Only members can broadcast to a group (closed groups)

The Amoeba
Reliable Broadcast Protocol

- Reliable broadcasting forms the basis for group communication in Amoeba



1. User process traps to kernel, passing it the message
2. Kernel accepts message, blocks user process
3. Kernel sends point-to-point message to sequencer
4. Sequencer allocates seq. number, broadcasts message with sequence number
5. Kernel sees message and unblocks sender

Reliable Broadcast

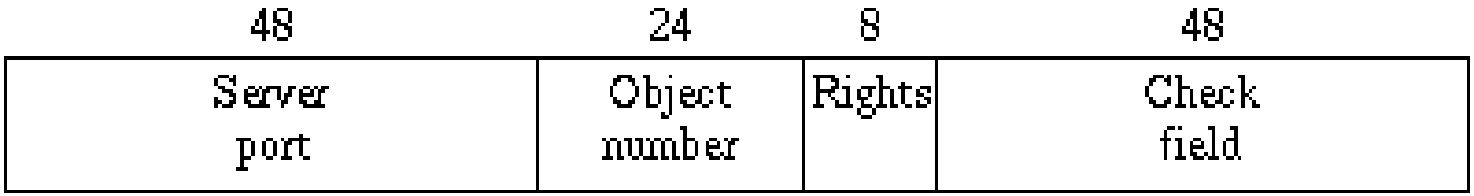
- Some receivers maintain a history buffer
- Re-transmissions are requested if message with unexpected sequence number arrives
  - Sequencer may retrieve message from history buffer
  - In case of sequencer crash, an alternative history buffer has to be exploited
- Leadership election protocol is employed to elect new sequencer in case of crashed sequencer
  - Depending on number of machines maintaining history buffers, varying numbers of crash faults can be tolerated

Objects and Capabilities

- Amoeba is an object-based system.
  - The system can be viewed as a collection of objects, on each of which there is a set of operations that can be performed.
  - The list of allowed operations is defined by the person who designs the object and who writes the code to implement it.
- Both hardware and software objects exist.
- Associated with each object is a capability
  - a kind of ticket or key that allows the holder of the capability to perform some (not necessarily all) operations on that object.
- Capabilities are protected cryptographically to prevent users from tampering with them.

Structure of a Capability

- A capability is 128 bits long and contains four fields.
- The first field is the server port , and is used to identify the (server) process that manages the object.
  - It is in effect a 48-bit random number chosen by the server.
- The second field is the object number ,
  - used by the server to identify which of its objects is being addressed.
  - Together, the server port and object number uniquely identify the object on which the operation is to be performed.
- The third field is the rights field, which contains a bit map telling which operations the holder of the capability may perform.
  - Since the operations are usually coarse grained, 8 bits is sufficient.

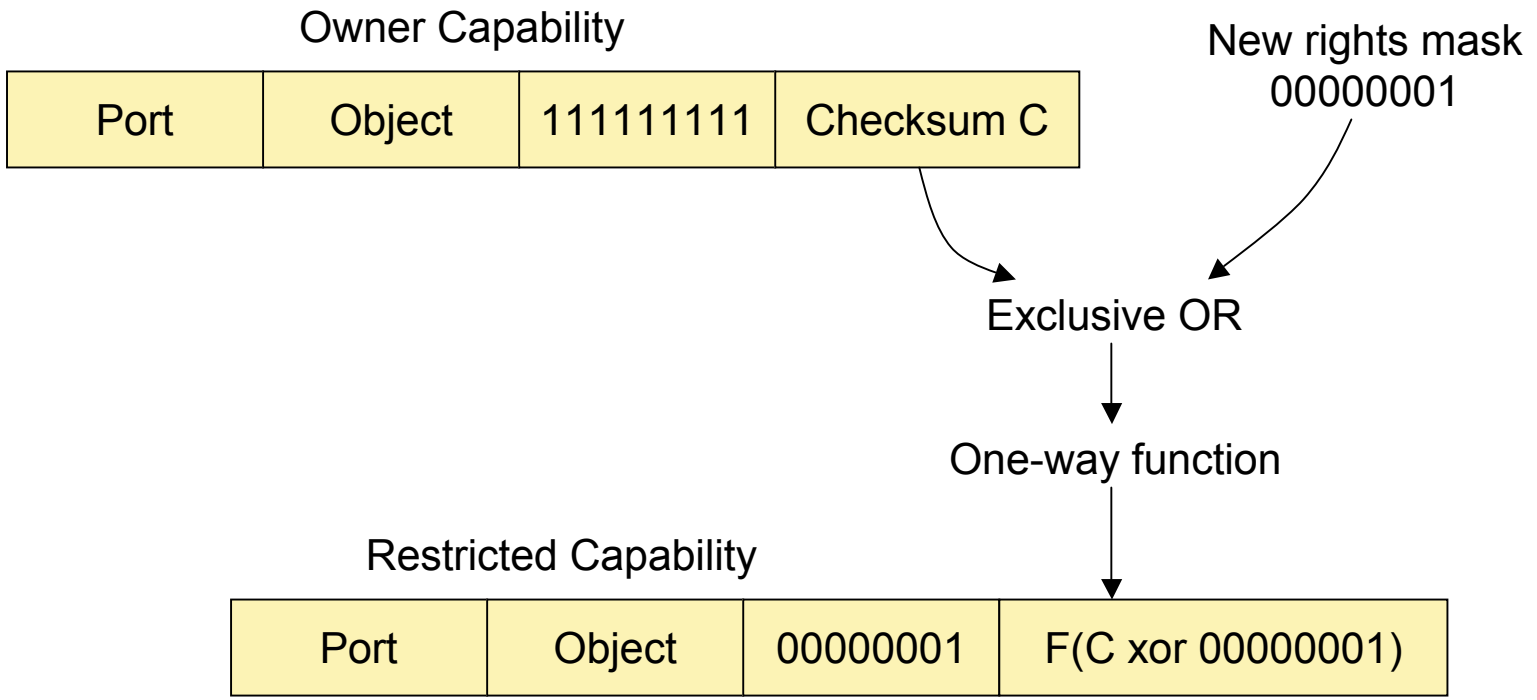


Objects and Capabilities (contd.)

- Each user process owns some collection of capabilities, which together define the set of objects it may access and the type of operations he may perform on each.
  - Capabilities are a unified mechanism for naming/accessing/protecting objects.
  - Function of the OS is to create an environment in which objects can be created and manipulated in a protected way.
- This object-based model is implemented using RPC
  - Remote procedure call
- Associated with each object is a server process.
  - To perform an operation on an object, a process sends a request message to the server that manages the object.
  - The message contains the capability for the object, a specification of the operation to be performed, and any parameters the operation requires.

Restricted Capabilities

- Generation of a restricted capability from an owner capability

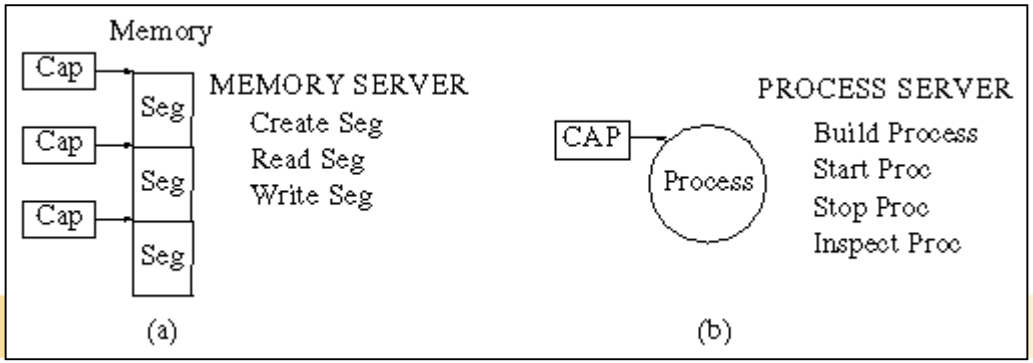


Standard Operations on Objects

Call	Description
Age	Perform a garbage collection cycle
Copy	Duplicate the object and return a capability for the copy
Destroy	Destroy an object and reclaim its storage
Getparams	Get parameters associated with the server
Info	Get an ASCII string briefly describing the object
Restrict	Produce a new, restricted capability for the object
Setparams	Set parameters associated with the server
Status	Get current status information from the server
Touch	Pretend the object was just used

Usage of Capabilities

- In (a), a group of three memory segments have been created, each of which has its own capability.
  - Capabilities, the creator can read and write the segments.
  - Given a collection of memory segments, a process can go to the process server and ask for a process to be constructed from them, as shown in (b).
  - This results in a process capability, through which the new process can be run, stopped, inspected, and so on.
- Location transparent and more efficient in a distributed system than the UNIX fork system call.



The Amoeba Servers

- Most of the traditional operating system services (such as the directory server) in Amoeba are implemented as server processes.
  - All these servers follow a common (message-based) communication model and use capabilities for access control
- Standard servers:
  - File server (bullet server)
  - Directory server (soap server)
  - Process server
  - Server for object replication
  - Monitoring servers for failure handling

The Bullet Server

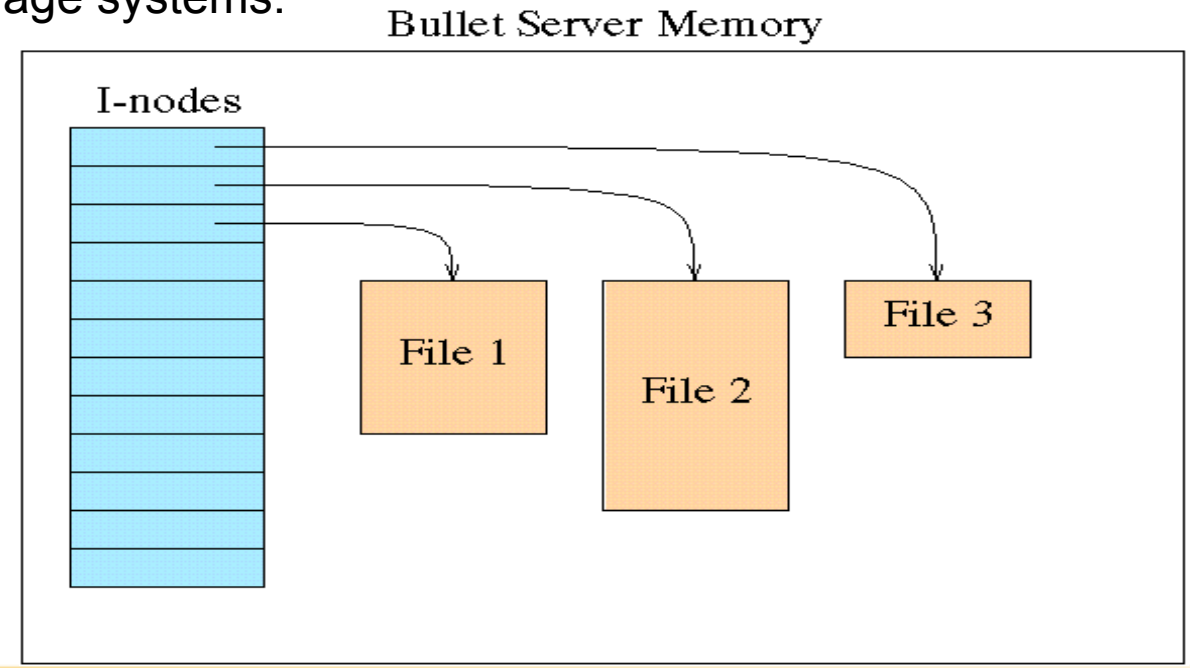
- Like all operating systems, Amoeba has a file system.
  - The choice of file system is not dictated by the operating system.
- The file system runs as a collection of server processes.
- The bullet server was designed to be very fast (hence the name).
  - Files are immutable
  - Once a file has been created, it cannot subsequently be changed.
  - It can be deleted, and a new file created in its place, but the new file has a different capability than the old one.
- This fact simplifies automatic replication
  - There are only two major operations on files: CREATE and READ.

The Bullet Server (contd.)

- Because files cannot be modified after their creation, the size of a file is always known at creation time.
  - Files can be stored contiguously on the disk, and also in the in-core cache.
  - By storing files contiguously, they can be read into memory in a single disk operation, and they can be sent to users in a single RPC reply message.
  - These simplifications lead to high performance.
- The bullet server maintains a table with one entry per file, analogous to the UNIX i-node table.
  - When a client process wants to read a file, it sends the capability for the file to the bullet server.
  - The server extracts the object number from the capability and uses it as an index into the in-core i-node table to locate the entry for the file.
  - The entry contains the random number used in the Check Field as well as some accounting information and two pointers: one giving the disk address of the file and one giving the cache address (if the file is in the cache).

Bullet Server Operation

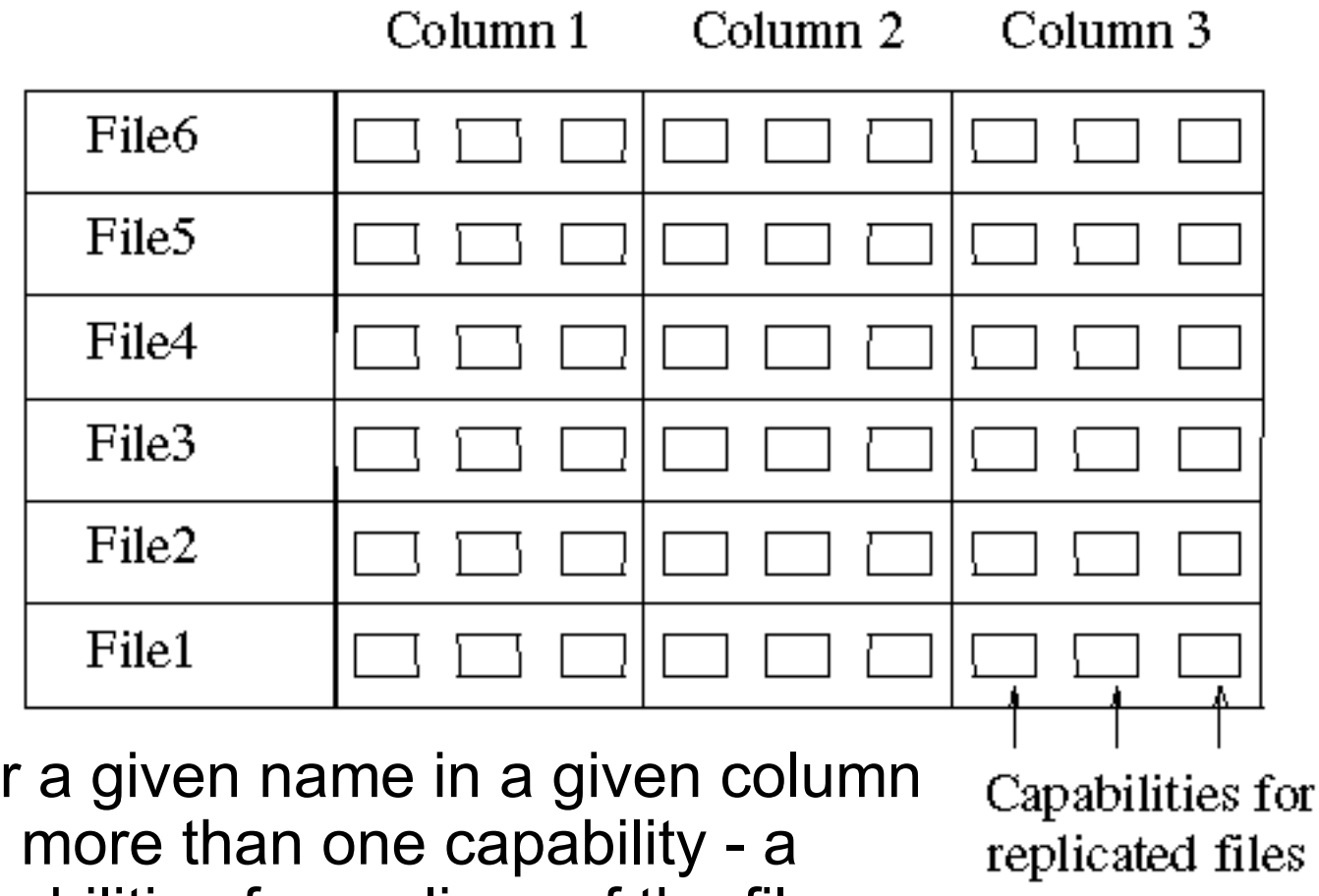
- Simple implementation leads to high performance.
  - Implementation is well suited to optical juke boxes and other write_once media, and can be used as a base for more sophisticated storage systems.



The Directory Server

- Provides a mapping from human-readable (ASCII) names to capabilities.
  - Users can create one or more directories, each of which contains multiple (name, capability-set) pairs.
  - Operations are provided to create and delete directories, add and delete (name, capability-set) pairs, and look up names in directories.
  - Unlike bullet files, directories are not immutable.
  - Entries can be added to existing directories and entries can be deleted from existing directories.

Sample Directory Layout



The entry for a given name in a given column may contain more than one capability - a group of capabilities for replicas of the file.

Sample Directory Layout (contd.)

- One row for each of the six file names stored in it.
- Three columns, each one representing a different protection domain.
  - First column might store capabilities for the owner (with all the rights bits on),
  - the second might store capabilities for members of the owner's group (with some of the rights bits turned off),
  - and the third might store capabilities for everyone else (with only the read bit turned on).
- When the owner of a directory gives away a capability for it, the capability is really a capability for a single column, not for the directory as a whole.

The Boot Server

- The boot server is used to provide a degree of fault tolerance to Amoeba.
- A process that is interested in surviving crashes can register with the boot server.
  - They agree on how often the boot server should poll, what it should send, and what reply it should get.
  - If the server responds correctly, the boot server takes no further action.
- If the server should fail to respond after a specified number of attempts,
  - the boot server declares it dead, and
  - arranges to allocate a new pool processor on which a new copy is started.

Amoeba as a Program Development Environment

- Partial UNIX emulation library
- Numerous UNIX application programs have been written or ported
  - binary compatibility with UNIX was not a goal
  - a set of library procedures emulate most of the common UNIX system calls, such as OPEN, READ, WRITE, CLOSE, FORK, and so on.
  - The ultimate goal is POSIX P1003.1 conformance, although that has not yet been achieved.
  - Each library procedure performs its work by making calls on the Amoeba servers.
- For example, the usual way the file system calls are handled is
  - to read the file into the caller's address space in its entirety (if possible),
  - operate on it locally, and then
  - write it back to the bullet server in a single CREATE operation.
  - Finally, the new capability is installed in the proper directory, removing the old one.
  - Then the old file can be deleted by the bullet server.

Programming with Amoeba (contd.)

- UNIX file system sits on top of the bullet server,
  - the latter's file replication facility is automatically present
- Amoeba and its servers are largely stateless,
  - Various aspects of UNIX require maintaining state information.
  - Session server keeps track of the the current UNIX login session.
  - Many UNIX-like utilities are available with Amoeba (well over 100).
  - Compilers are available for C, Pascal, Modula 2, Orca.
- Amoeba programs contain any UNIX code whatsoever. AT&T license is not required for Amoeba.
- Amoeba is distributed with full source code.

Orca - Parallel Programming

- Another use of Amoeba is for supporting parallel programming.
  - the processor pool model is an attractive way to exploit massive parallelism.
  - processors communicate using RPC and other techniques.
- Orca (Bal, 1990; Bal and Tanenbaum, 1991
- Orca is based on shared data objects upon which specific operations are defined (in effect, abstract data types).
  - Processes on different processors can share these abstract objects, even though the system itself does not necessarily contain any physical shared memory.
  - How this illusion is supported is the job of Amoeba and the Orca run-time system.
  - The basic idea is that each shared object is replicated in full on all processors that are running a process interested in the shared object (Bal and Tanenbaum, 1991).

Orca on Amoeba

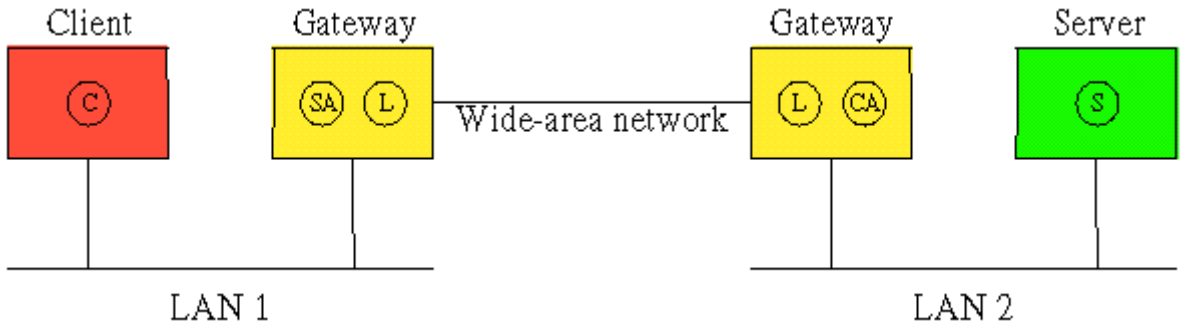
- Orca operations are divided into two categories:
  - those involving only reading the object, and those changing the object.
- The read operations are easy.
  - Since a copy of the object is located on each machine, the operation can be performed entirely locally, with no network traffic.
  - This means that they can be performed with no delay, highly efficiently.
- Operations that involve changing an object are more complicated.
  - The algorithm used is based on one of the services offered by Amoeba 5.0, reliable broadcast (Kaashoek and Tanenbaum, 1991).
  - By this we mean, a message from one sender can be sent to a group of receivers with certainty that all of them will receive it (unless some of them crash).

Amoeba on a Wide Area Network

- Main difference between a LAN and a WAN is the lack of broadcasting on a WAN.
  - When a process performs an RPC using a capability whose port has not previously been used, the kernel on that machine locates the destination by broadcasting a special LOCATE packet.
  - On a wide area network, such broadcasts are not possible,
  - A slightly different approach is taken, one that preserves the goal of transparency---the client cannot tell where the server is,
  - all actions taken by both client and server are the same, whether they are on the same network or not.
- WAN services are required to publish their port.
  - Publishing a port is done by the (human) owner of the service,
  - To publish the port, the owner runs a special program that sends the server's port and network address to the set of gateways on whose networks the server is to be known.

Amoeba on a WAN (contd.)

- When a WAN service is published, a special server agent process is created on the gateway machine.
  - This server agent listens to the server's port.
  - When a client on the server agent's LAN does an RPC to the server, the client's kernel broadcasts a LOCATE packet, which is received by the gateway.
  - The gateway's kernel responds in the usual way,
  - The server agent then passes it to a link process, which transmits it over the wide-area link using whatever protocol is required there.
- At the destination, a client agent is created, which does an RPC with the server.
  - The reply follows the reverse path back to the client.



Lessons learned

- The use of a microkernel has been very satisfactory.
  - A microkernel-based design is simple and flexible.
  - The potential fear that it would be too slow for practical use has not been borne out.
  - By making it possible to have servers that run as user processes, we have been able to easily experiment with different kinds of servers and to write and debug them without having to bring down and reboot the kernel all the time.
  - For the most part, RPC communication has been satisfactory, as have the three primitives for accessing it.
  - Occasionally, however, RPC gives problems in situations in which there is no clear master-slave relation, such as in UNIX pipelines (Tanenbaum and van Renesse, 1988).
  - Another difficulty is the fact that although RPC is fine for sending a message from one sender to one receiver, it is less suited for group communication.

Lessons learned (contd.)

- The object-based model for services has also worked well.
  - Clear model for the design of services.
- The use of capabilities for transparent naming and protection can be largely regarded as a success.
  - It is conceivable, however, that if the system were to be extended to millions of machines worldwide, the idea of using capabilities would have to be revisited.
  - The fear is that casual users might be too lax about protecting their capabilities.

Additional Reading

- Tanenbaum, A.S., Kaashoek, M.F., Renesse, R. van, and Bal, H.: "The Amoeba Distributed Operating System-A Status Report," Computer Communications, vol. 14, pp. 324-335, July/August 1991.
- Tanenbaum, A.S., Renesse, R. van, Staveren, H. van., Sharp, G.J., Mullender, S.J., Jansen, A.J., and Rossum, G. van: "Experiences with the Amoeba Distributed Operating System," Commun. ACM, vol. 33, pp. 46-63, Dec. 1990.
- Kaashoek, M.F., and Tanenbaum, A.S.: "Efficient Reliable Group Communication for Distributed Systems" (submitted for publication, 1994)
- Tanenbaum, A.S., „Distributed Operating Systems“, Prentice-Hall, 1995.
- http://www.cs.vu.nl/vakgroepen/cs/amoeba_papers.html