AMOEBA – A DISTRIBUTED OPERATING SYSTEM

A Case Study by:

R. Yamini

Dept. of CA,
SRM University,
Kattankulathur
Overview of AMOEBA
Introduction

• What is amoeba?
• The Amoeba distributed operating system project is a research effort aimed at understanding how to connect computers together in a seamless way.
A Distributed OS

• What is a distributed OS?
• In this, users effectively log into the system as a whole, and not to a specific machine.
• When a program is run, the system, not the user, decides the best place to run it.
Introduction to Amoeba

• Amoeba is an OS that performs all the standard functions of any OS, but it performs them with a collection of machines.
• One of the main goals of the Amoeba development team was to design a transparent distributed system that allows users to log into the system as a whole.
• When a user logs into an Amoeba system, it seems like a powerful, single-processor, time-sharing system.
Its origin

- Amoeba was originally designed and implemented at the Vrije University in Amsterdam (the Netherlands) under the direction of Professor Andrew S. Tanenbaum.
- Now, it is being jointly developed there and at the Center for Mathematics and Computer Science, also in Amsterdam.
- Four basic design goals were apparent in Amoeba: Distribution, Parallelism, Transparency, and Performance.
It is …

• Amoeba is a distributed system that allows several machines connected over a network to operate as a single system.
• The machines that make an Amoeba kernel can be spread throughout a building on a Local Area network (LAN). It only provides limited support for Wide Area Network (WAN) connections
The goal

• The first goal of the design team was to make Amoeba give its users the illusion of interacting with a single system, even though the system was distributed.

• In addition to managing the Amoeba network, an Amoeba system can also act as a router to connect several other networks together.

• This is all accomplished with a (newly developed) High Performance network protocol called FLIP (Fast Local Internet Protocol).
Features

- Amoeba is also a parallel system.
- On an Amoeba system, a single program or command can use multiple processors to increase performance.
- Special development tools have been developed for an Amoeba environment that take advantage of the inherent parallelism.
- When a user logs into the Amoeba system that they can access the entire system, and are not limited to only operations on their home machine.
Features...

• The Amoeba architecture is designed as a collection of micro-kernels.

• Amoeba implements a standard distributed client / server model, where user processes and applications (the clients) communicate with servers that perform the kernel operations.

• An Amoeba system consists of four principle components: user workstations, pool processors, specialized servers, and gateways.
AMOEBA ARCHITECTURE

• The workstations allow the users to gain access to the Amoeba system.
• There is typically one workstation per user, and the workstations are all diskless, so they act as intelligent terminals.
• Amoeba supports X-windows and UNIX emulation, so X-window terminals are often used to facilitate interaction with UNIX systems.
Pool Processor

• The pool processors are unique with Amoeba.
• This is a group of CPUs that can be dynamically allocated as needed by the system, and are returned to the pool when processing is complete.
Servers & Communication

- At the heart of the Amoeba system are several specialized servers that carry out and synchronize the fundamental operations of the kernel.
- The Bullet Server, the Directory Server, the Replication server and the Run server are different servers.
- Remote Procedure call and Group communication are two different kinds of communication procedures in amoeba.
OBJECTS IN AMOEBA
OBJECTS

- CONCEPT OF SERVER PROCESSES
- CONCEPT OF OBJECTS
- OBJECTS ARE PASSIVE
- OBJECTS ARE MANAGED BY SERVER PROCESSES
**CAPABILITIES**

- **THE FORMAT OF CAPABILITY**

<table>
<thead>
<tr>
<th>Server Port(8)</th>
<th>Object(3)</th>
<th>Rights(1)</th>
<th>Check(8)</th>
</tr>
</thead>
</table>

- **THE PORT** is the logical address where the server can be found.
- **THE OBJECT** field is used by the server to identify the specific object.
- **THE RIGHTS** field tells which of the operations the holder may perform.
- **THE CHECK** field is used for validating the capability.
OBJECT PROTECTION

• WHEN OBJECT IS CREATED SERVER PICKS UP A RANDOM CHECK FIELD
• THE RANDOM NUMBER IS STORED BOTH IN SERVER’S OWN TABLES AND CHECK FIELD OF CAPABILITY
• ALL THE RIGHTS BITS IN A NEW CAPABILITY ARE INITIALLY ON
• WHEN THE CAPABILITY IS SENT BACK TO THE SERVER IN A REQUEST TO PERFORM SOME OPERATION, CHECK FIELD IS VERIFIED
Generation of restricted capability from an owner capability

Owner Capability

<table>
<thead>
<tr>
<th>Server Port(8)</th>
<th>Object(3)</th>
<th>11111111</th>
<th>C</th>
</tr>
</thead>
</table>

New rights mask
00000001

Exclusive OR

One way function

| Server Port(8) | Object(3) | 00000001 | f(C XOR 00000001) |
PROCESS MANAGEMENT
PROCESS MANAGEMENT IN AMOEBA

- Processes.
- Process management.
- Process descriptor.
- Library procedures.
- Threads.
- Synchronization between threads.
• A *process* is an object in amoeba.
• Process creation in amoeba is different from *unix*.
• In amoeba it is possible to create a new process on a specified processor with the intended memory image starting at the beginning.
• In this one aspect a process creation is similar to MS-DOS.
• A process can be in two states.
  1. Running
  2. Stunned
PROCESS MANAGEMENT

- Process management is handled at 3 levels in amoeba.
  1. The lowest level are the process servers, which are kernel threads running on every machine.
  2. We have a set of library procedures that provide a more convenient interface for user programs.
  3. Finally, the simplest way to create a new process is to use the run server, which does the most of the work of determining where to run the new process.
Some of the process management calls use a data structure called process descriptor to provide information about the process to be run.

1. First field, *host descriptor* tells which CPU architecture the process can run on.
2. Second field contains a capability for communicating the exit status to the owner.
3. Third field *memory map* contains a descriptor for each segment in process’ address space.
4. Finally the process descriptor also contains a *thread descriptor* for each thread in the process.
Process descriptor

Architecture = 386

Capability for exit status

Segment Descriptors

<table>
<thead>
<tr>
<th>Th1</th>
<th>Th2</th>
<th>Th3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC1</td>
<td>PC2</td>
<td>PC3</td>
</tr>
<tr>
<td>SP1</td>
<td>SP2</td>
<td>SP3</td>
</tr>
</tbody>
</table>

Program counters

Text

Shared Data

Private Data

Stacks

Segments
LIBRARY PROCEDURES

- Low level process interface consists of library procedures.
  
  1. *Exec*: It’s function is to do RPC with the specified process server asking to run the process.
  
  2. *Getload*: It returns the information about the CPU speed, current load and the amount of memory free at the moment.
  
  3. *Stun*: The parent can suspend a process by stunning it.
THREADS

- Amoeba supports a simple threads model.
- Glocal variables.
SYNCHRONIZATION BETWEEN THREADS

- Three methods are provided for threads to synchronize.
  1. Signals.
  2. Mutexes.
SYNCHRONIZATION METHODS

• *Signals* are asynchronous interrupts sent from one thread to another thread in the same process.

• *Mutex* is like a binary semaphore. It can be in one of the two states, locked or unlocked.

• *Semaphores* are slower than mutexes but there are times when they are needed.
• All threads are managed by the kernel. The advantage of this design is that when a thread does RPC, the kernel can block the thread and schedule another one in the same process if one is ready.

• Thread scheduling is done using priorities, with kernel threads getting higher priority than user threads.
MEMORY MANAGEMENT
MEMORY MANAGEMENT

• Entirely in physical memory.
• Segments contiguous in address space.
• Segments are objects.
• Any process with capability to segment could read/write it (with proper permissions).
• Shared memory communications (need not be on same machine).
• Main memory file server.
SEGMENTS

• A segment is a contiguous block of memory that can contain code or data.
• Each segment has a capability that permits its holder to perform operations on int, such as reading and writing.
• A segment is somewhat like an in-core file, with similar properties.
• Segments cannot be swapped or paged.
• When a segment is created it is given an initial value. This size may change during the process execution.
• The segment may also be given initial value either from another segment or from a file.
• Processes have several calls available to them for managing segments.
• Most important among these is the ability to create, read and write segments.
• When a segment is created, the caller gets back a capability for it.
• This capability is used for all other calls involving the segment.
A main memory file server can be constructed using segments because of their read and write properties. The server creates a segment as large as it can. This segment will be used as a simulated disk. The server then formats the segment as a file system, putting in whatever data structures it needs to keep track of the files. After that, it is open for business, accepting and processing requests from clients.
MEMORY MODEL

• Amoeba has an extremely simple memory model.
• A process can have any number of segments.
• They can be located wherever it wants in the process’s virtual address space.
• The virtual address spaces in Amoeba are constructed from segments.
• Segments are not swapped or paged, so a process must be entirely memory resident to run.
• Although the hardware MMU is used, each segment is stored contiguously in memory.

Cont…. 
• This design was done for three reasons: Performance, Simplicity, Economics.

• Having a process entirely in memory all the time makes RPC go faster.

• This design has allowed Amoeba to achieve extremely high transfer rates for large RPC’s.

• Not having paging or swapping makes the system considerably simpler and makes the kernel smaller and more feasible.

• Memory is becoming so cheap that within a few years, all Amoeba machines will probably have tens of megabytes of it.
MAPPED SEGMENTS

• When a process is started, it must have at least one segment.
• Once it is running, a process can create additional segments and map them into its address space at any unused virtual address.
A process with three segments mapped into its virtual address space.
UNMAPPED SEGMENTS

• A process can also unmap segments.
• A process can specify a range of virtual addresses and request that the range be unmapped.
• When a segment is unmapped a capability is returned, so the segment may still be accessed, or even be remapped.
COMMUNICATIONS IN AMOEBA
COMMUNICATION IN AMOEBA

- Remote procedure call (RPC)
- Group communication
REMOTE PROCEDURE CALL

• POINT-TO-POINT COMMUNICATION
  With client sending a message followed by server sending a reply

• REQUEST/REPLY MESSAGE EXCHANGE
  Each standard server defines a procedural interface that clients can call
REMOTE PROCEDURE CALL

- FINDING SERVER’S ADDRESS

Addressing is done by allowing any thread to choose a random 48-bit number, called a PORT.
RPC Primitives

- The RPC mechanism makes use of three principal kernel primitives:
  1. `get_request` - indicates server’s willingness to listen to port
  2. `put_reply` - done by a server when it has a reply to send
  3. `trans` - send a message from client to server and wait for reply
Usage of sys-calls

- `get_request(&header, buffer, bytes)`
- `read(fd, buffer, bytes)`
- `put-reply(&header, buffer, bytes)`
- `trans(&header1, buffer1, bytes1, &header2, buffer2, bytes2)`
<table>
<thead>
<tr>
<th>Port</th>
<th>6 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>6 bytes</td>
</tr>
<tr>
<td>Private part</td>
<td>10 bytes</td>
</tr>
<tr>
<td>Command</td>
<td>2 bytes</td>
</tr>
<tr>
<td>Offset</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Size</td>
<td>2 bytes</td>
</tr>
<tr>
<td>Extra</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>
SECURITY

• It is possible for an intruder to impersonate a server by just doing a \textit{get\_request} on server’s port. Amoeba solves this problem cryptographically.

• Each port is a pair of ports:
  • get-port (private, only to server)
  • Put-fort (public, known to world)

• \( \text{put-port}=f(\text{get-port}) \)
• \( f \)- one way function
RELATIONSHIP BETWEEN GET-PORTS AND PUT-PORTS

Client → trans

Kernel

Server → get_request

Network

Packets contain put-port

Table of ports being listened to holds put-ports
Properties of RPC Mechanism

• It supports only synchronous type of data
• Messages are unbuffered. Message is simply discarded if the receiver is not in a ready state to receive it. The sending kernel will time out and retransmit the message. Flexibility is provided to specify the maximum retransmissions, after which the kernel should give up and report failure
Properties of RPC Mechanism

• It supports at-most-once semantics. that is the system guarantees that an RPC will never be carried out more than once, even if the server crashes and is rapidly rebooted.

• Stateless servers are used. therefore, each RPC is completely self-contained and does not depend on any previous information stored in the server’s memories.
GROUP COMMUNICATION

• RPC is not the only form of communication supported by amoeba. It also supports group communication.
• Amoeba uses the concepts of closed groups.
• Processes can join and leave the group dynamically and can members of multiple group at the same time.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Object</td>
<td>Create a new group and set its parameters</td>
</tr>
<tr>
<td>Join Group</td>
<td>Make the caller a member of a group</td>
</tr>
<tr>
<td>Leave Group</td>
<td>Remove the caller from a group</td>
</tr>
<tr>
<td>Send To Group</td>
<td>Reliably send a message to all members of a group</td>
</tr>
<tr>
<td>Receive From Group</td>
<td>Block until a message arrives from a group</td>
</tr>
<tr>
<td>Reset Group</td>
<td>Initiate recovery after a process crash</td>
</tr>
</tbody>
</table>

**Amoeba Group Communication Primitives**
Amoeba ensures ordered delivery of messages.
A sequencer process is used for properly sequencing the messages received by a group.
It is chosen by using an *Election Algorithm*. 
PROPERTIES OF GROUP COMMUNICATION

• Amoeba ensures reliable delivery of messages.
• The basic mechanisms used to ensure reliable message delivery are timed out based re-transmissions.
• Use of any message identifiers to detect duplicate messages.
Properties of group communication mechanism

• It ensures ordered delivery of messages. That is if two processes send messages to a group almost simultaneously, the system ensures that all group members will receive the messages in same order.
PROPERTIES OF GROUP COMMUNICATION

• Amoeba can withstand the loss of an arbitrary collection of $k$ (the degree of resilience) processes.
• $k$ is specified by user as a parameter for creating a group.
• The larger the value of $k$, more the redundancy is required and slower the group communication becomes.
THE AMOEBA SERVERS
THE AMOEBA SERVERS

- The Amoeba kernel essentially handles communication and some process management, and little else.
- The kernel takes care of sending and receiving messages, scheduling processes, and some low-level memory management. Everything else is done by user processes.
**Stub Processes…**

- All standard servers in Amoeba are defined by a set of stub processes.
- Stubs precisely define what the server provides and what their parameters are.
- The older stubs are written in C whereas newer ones are defined in AIL, Amoeba Interface Language.
Different Servers...

- The standard file system consists of three servers.
  1. The Bullet server
  2. The Directory server
  3. The Replication server
The File Server...

- The file server is a user process.
- Also called as Bullet Server.
- It is a simple file server that maintains immutable files.
- It stores files contiguously on disk and in cache.
- Supported operations: create file, read file, delete file
The Bullet Server...

- When creating a file, the entire file data is provided and the file's capability is returned.
- To modify this file, the client sends back the capability through a RPC.
- A part of the file can also be read by sending offset and a byte count.
- Thus, aims at the convenience of the client.
Types of files

- The two kinds of files are:
  1. Committed
  2. Uncommitted
- Uncommitted files can be changed whereas committed ones are permanent.
- Only committed files can be read.
# The Bullet Server Interface

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create a new file; optionally commit it as well</td>
</tr>
<tr>
<td>Read</td>
<td>Read all parts of a specified file</td>
</tr>
<tr>
<td>Size</td>
<td>Return the size of a specified file</td>
</tr>
<tr>
<td>Modify</td>
<td>Overwrite n bytes of an uncommitted file</td>
</tr>
<tr>
<td>Insert</td>
<td>Insert or append n bytes to an uncommitted file</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete n bytes from an uncommitted file</td>
</tr>
</tbody>
</table>
Timeouts

• If a file is created and its capability is lost, the file can never be accessed.
• An uncommitted file will be deleted if it is not accessed for 10 minutes.
• Age and touch operations…
• Age operation runs through the server and goes on decrementing MAX_LIFETIME and deletes when it becomes 0.
• Every file is touched once in an hour and is deleted if it is not accessed for 24 hours.
• This mechanism removes lost files.
• Fragmentation problem.
The Directory Server

• The Directory server allows one to use names, by providing mapping from user names onto capabilities.
• Directories are also objects protected by capabilities.
• Entries in the directory may be of different varieties.
• A directory may return a capability to another directory.
The Directory Server...

• The directory usually has five entries – ASCII string, Capability set, Owner, Group and Others.
• The rights, as in UNIX, are specified for three different user groups. The directory actually creates a new capability with the specified rights and returns that to the user.
• There is no concept of a single, global root directory.
• Directories like bin, dev, etc, public (cap, share, pool).
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>CREATE A NEW DIRECTORY</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE A DIRECTORY OR AN ENTRY IN A DIRECTORY</td>
</tr>
<tr>
<td>APPEND</td>
<td>ADD A NEW DIRECTORY ENTRY TO A SPECIFIED DIRECTORY</td>
</tr>
<tr>
<td>REPLACE</td>
<td>REPLACE A SINGLE DIRECTORY ENTRY</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>RETURN THE CAPABILITY SET CORRESPONDING TO A SPECIFIED NAME</td>
</tr>
<tr>
<td>GETMASKS</td>
<td>RETURN THE RIGHTS MASKS FOR THE SPECIFIED ENTRY</td>
</tr>
<tr>
<td>CHMOD</td>
<td>CHANGE THE RIGHTS BITS IN AN EXISTING DIRECTORY ENTRY</td>
</tr>
</tbody>
</table>
The Replication Server

- Objects managed by directory server can be replicated automatically by the replication server.
- Lazy replication
- Initially one copy of object is created and later replication server is invoked to produce identical replicas.
- It sends age messages.
Other Servers

- Some other servers are – The Run Server, The Boot Server and The TCP/IP server.
- A run server manages one or more processor pool. A processor pool is represented by a pooldir.
- The boot server checks whether all other servers are running smoothly or not.
- Communication with x-server is also possible.
Wide Area Amoeba

- Amoeba was designed with the idea that a collection of machines on a LAN would be able to communicate over a wide-area network with a similar collection of remote machines.

- The primary goal of the wide area-networking in Amoeba has been to achieve transparency without sacrificing performance.

- Broadcast over WAN is both cost ineffective and for some services (e.g., print server) of no use over a WAN. These problems are solved by introducing the concept of *publishing*. 
Conclusion

• On the whole, the idea of an object-based system has worked well.
• In future, Amoeba will support 256-bit capabilities and will have a room for a location hint which can be exploited by SWAN servers for locating objects in the WAN.
• RPC communication is excellent. The only problem being lack of group communication support.
• An improvement that developers of Amoeba will be making in Amoeba's memory and process management is to allow for preemption of threads, which now use the run-to-completion scheduling semantics.
Conclusion…

• The design of the file server and directory server has been very successful. The separation of the file server and directory server is an important aspect of this design.

• Internetworking has been handled in a very innovative manner with good results.

• Amoeba has also been used for parallel computing, although designed for distributed computing.

• The processor pool has been used to achieve large speed ups on a single problem. To program these parallel applications, a new language Orca has been under development.
Present Status

- Currently, amoeba is being used for several practical situations from a development environment, for parallel computing, to industrial applications.
- The RIT Computer Science House is implementing amoeba. Details can be got about it from www.csh.rit.edu
- Amoeba runs on Sun SPARC stations, Sun 3/60, and 3/50 workstations, Intel 86/486/Pentium/Pentium Pro, and 68030 VME-bus boards.
- More information about the present status can be got at: www.am.cs.vu.nl