The Amoeba Distributed Operating System-Case Study

Harshada Dhande
IT department
Vidyalankar Institute of Technology
Mumbai University.
harshudhande@gmail.com

***Abstract*—This document gives information about Amoeba Operating system with all the features provided by this technology. The software and hardware used with the special purposes so that this operating system was designed is also mentioned in the paper.**

I. INTRODUCTION

Roughly speaking, we can divide the history of modern computing into the following eras:

- 1970s: Timesharing (1 computer with many users)
- 1980s: Personal computing (1 computer per user)
- 1990s: Parallel computing (many computers per user)

Until about 1980, computers were huge, expensive, and located in computer centers. Most organizations had a single large machine.In the 1980s, prices came down to the point where each user could have his or her own personal computer or work station. These machines were often networked together,so that users could do remote logins on other people's computers or share files in various (often ad hoc) ways. Nowadays some systems have many processors per user, either in the form of a parallel computer or a large collection of CPUs shared by a small user community. Such systems are usually called *parallel* or *distributed computer systems*.This development raises the question of what kind of software will be needed for these new systems. The answer of this question has resulted in the development of a new distributed operating system, called *Amoeba*, designed for an environment consisting of a large number of computers. Amoeba is available for free to universities and other educational institutions and for special commercial prices and conditions to corporate, government, and other users, as described later.Section 2 describes actually what is Amoeba. Section 3explain design goals. Section 3 and 4 talks about the structure of Amoeba with some concepts providing by it. Section 5 and 6 explains technical and nontechnical aspects.

II. WHAT IS AMOEBA?

Amoeba is a general-purpose distributed operating system. It is designed to take a collection of machines and make them act together as a single integrated system. In general, users are not aware of the number and location of the processors that run their commands, nor of the number and location of the file servers that store their files. To the casual user, an Amoeba system looks like a single old-fashioned time-sharing system. Amoeba is an ongoing research project. It should be thought of as a platform for doing research and development in distributed and parallel systems, languages, protocols and applications. Although it provides some UNIX emulation, and has a definite UNIX-like flavor (including over 100 UNIX-like utilities), it is NOT a plug-compatible replacement for UNIX. It should be of interest to educators and researchers who want the source code of a distributed operating system to inspect and tinker with, as well as to those who need a base to run distributed and parallel applications. Amoeba is intended for both ''distributed'' computing (multiple independent users working on different projects) and ''parallel'' computing (e.g., one user using 50 CPUs to play chess in parallel). Amoeba provides the necessary mechanism for doing both distributed and parallel applications, but the policy is entirely determined by user-level programs. For example, both a traditional (i.e. sequential) 'make' and a new parallel 'amake' are supplied.

III. DESIGN GOALS

The basic design goals of Amoeba are:

- Distribution—Connecting together many machines
- Parallelism—Allowing individual jobs to use multiple CPUs easily
- Transparency—Having the collection of computers act like a single system

- Performance—Achieving all of the above in an efficient manner

Amoeba is a *distributed* system, in which multiple machines can be connected together. These machines need not all be of the same kind. The machines can be spread around a building on a LAN. Amoeba uses the high performance FLIP network protocol for LAN communication. If an Amoeba machine has more than one network interface it will automatically act as a FLIP router between the various networks and thus connect the various LANs together. Amoeba is also a *parallel* system. This means that a single job or program can use multiple processors to gain speed. For example, a branch and bound problem such as the Traveling Salesman Problem can use tens or even hundreds of CPUs, if available, all working together to solve the problem more quickly. Large ‘‘back end’’ multiprocessors, for example, can be harnessed this way as big ‘‘compute engines.’’ Another key goal is *transparency*. The user need not know the number or the location of the CPUs, nor the place where the files are stored. Similarly, issues like file replication are handled largely automatically, without manual intervent- ion by the users. Put in different terms, a user does not log into a specific machine, but into the system as a whole. There is no concept of a ‘‘home machine.’’ Once logged in, the user does not have to give special *remote login* commands to take advantage of multiple processors or do special *remote mount* operations to access distant files. To the user, the whole system looks like a single conventional timesharing system. Performance and reliability are always key issues in operating systems, so substantial effort has gone into dealing with them. In particular, the basic communication mechanism has been optimized to allow messages to be sent and replies received with a minimum of delay, and to allow large blocks of data to be shipped from machine to machine at high bandwidth. These building blocks serve as the basis for implementing high performance subsystems and applications on Amoeba.

IV. SYSTEM ARCHITECTURE

Since distributed and parallel computing is different from personal computing, it is worthwhile first describing the kind of hardware configuration for which Amoeba was designed. A typical Amoeba system will consist of three functional classes of machines. First, each user has a workstation for running the user interface, the X window system. This workstation can be a typical engineering workstation, or a specialized X terminal. It is entirely dedicated to running the user interface, and does not have to do other computing. Second, there exists a pool of processors that are dynamically allocated to users as required. These processors can be part of a multiprocessor or multicomputer, be a collection of single-board computers or be a group of workstations allocated for this purpose. Usually, each pool processor has several megabytes of private memory, that is, pool processors need not have any shared memory (but it is not forbidden). Communication is performed by sending packets over the LAN. All the heavy computing happens in the processor pool. Third, there are specialized servers, such as file servers and directory servers that run all the time. They may run on processor pool processors, or on dedicated hardware ,as desired. All these components must be connected by a fast LAN. At present only Ethernet is supported, but ports to other LANs are possible.

V. FUNDAMENTAL CONCEPTS OF AMOEBA

A. *Microkernel+server architecure*

Amoeba was designed was with what is currently termed a *microkernel* architecture. This means that every machine in an Amoeba system runs a small, identical piece of software called the *kernel*. The kernel supports the basic process, communication, and object primitives. It also handles raw device I/O and memory management. Everything else is built on top of these fundamentals, usually by user-space *server* processes.

Thus the system is structured as a collection of independent processes. Some of these are user processes, running application programs. Such processes are called *clients*. Others are *server* processes, such as the Bullet file server or the directory server. The basic function of the microkernel is to provide an environment in which clients and servers can run and communicate with one another.

This modular design makes it easier to understand, maintain, and modify the system. For example, since the file server is an isolated server, rather than being an integral part of the operating system, it is possible for users to implement new file servers for specialized purposes (e.g. NFS, database). In conventional systems, such as UNIX, adding additional user-defined file systems is infeasible.

B. *Threads*

In many traditional operating systems, a process consists of an address space and a single thread of control. In Amoeba, each process has its own address space, but it may contain multiple ‘‘threads of control’’ (threads). Each thread has its own program counter and its own stack, but shares code and global data with all the other threads in its process.

Having multiple threads inside each process is convenient for many purposes and fits into the model of

distributed and parallel computing very well. For example, a file server may have multiple threads, each thread initially waiting for a request to come in. When a request comes in, it is accepted by some thread, which then begins processing it. If that thread subsequently blocks waiting for disk I/O, other threads can continue. Despite their independent control, however, all the threads can access a common block cache, using semaphores to provide inter-thread synchronization. This design makes programming servers and parallel applications much easier. Not only are user processes structured as collections of threads communicating by RPC, but the kernel is as well. In particular, threads in the kernel provide access to memory management services.

C. *Remote Procedure Call*

Threads often need to communicate with one another. Threads within a single process can just communicate via the shared memory, but threads located in different processes need a different mechanism. The basic Amoeba communication mechanism is the *remote procedure call* (RPC). Communication consists of a client thread sending a message to a server thread, then blocking until the server thread sends back a return message, at which time the client is unblocked.

To shield the naive user from these details, special library procedures, called *stubs*, are provided to access remote services. Amoeba has a special language called Amoeba *Interface Language* (AIL) for automatically generating these stub procedures. They marshal parameters and hide the details of the communication from the users.

D. *Group Communication*

For many applications, one-to-many communication is needed, in which a single sender wants to send a message to multiple receivers. For example, a group of cooperating servers may need to do this when a data structure is updated. It is also frequently needed for parallel programming. Amoeba provides a basic facility for reliable, totally-ordered group communication, in which all receivers are guaranteed to get all group messages in exactly the same order. This mechanism simplifies many distributed and parallel programming problems.

E. *Objects and Capabilities*

There are two fundamental concepts in Amoeba: objects and capabilities. All services and communication are built around them. An *object* is conceptually an abstract data type. That is, an object is a data structure on which certain operations are defined. For example, a directory is an object to which certain operations can be applied, such as ‘‘enter name’’ and ‘‘look up name.’’ Amoeba primarily supports software objects, but hardware objects also exist. Each object is managed by a server process to which RPCs can be sent. Each RPC specifies the object to be used, the operation to be performed, and any parameters to be passed. When an object is created, the server doing the creation constructs a 128-bit value called a *capability* and returns it to the caller. Subsequent operations on the object require the user to send its capability to the server to both specify the object and prove the user has permission to manipulate the object. Capabilities are protected cryptographically to prevent tampering. All objects in the entire system are named and protected using this one simple, transparent scheme.

F. *Memory Management*

The Amoeba memory model is simple and efficient. A process’ address space consists of one or more segments mapped onto user-specified virtual addresses. When a process is executing, all its segments are in memory. There is no swapping or paging at present, thus Amoeba can only run programs that fit in physical memory. The primary advantage of this scheme is simplicity and high performance. The primary disadvantage is that it is not possible to run programs larger than physical memory.

VI. SOFTWARE OUTSIDE THE KERNEL

The job of the Amoeba microkernel is to support threads, RPC, memory management and I/O. Everything else is built on top of these primitives.

A. *Bullet File Server*

The standard Amoeba file server has been designed for high performance and is called the *Bullet server*. It stores files contiguously on disk, and caches whole files contiguously in core. Except for very large files, when a user program needs a file, it will request that the Bullet server send it the entire file in a single RPC. A dedicated machine with at least 16 MB of RAM is needed for the Bullet file server for installation (except on the Sun 3 where there is a maximum of 12 MB). The more RAM the better, in fact. The performance is improved with a larger file cache. The maximum file size is also limited by the amount of physical memory available to the Bullet server.

B. *Directory Server*

In contrast to most other operating systems file management and file naming are separated in Amoeba. The Bullet server just manages files, but does not handle naming. It simply reads and writes files,

specified by capabilities. A capability can be thought of as a kind of handle for an object, such as a file. A directory server maps ASCII strings onto capabilities. Directories contain (ASCII string, capability) pairs; these capabilities will be for files, directories, and other objects. Since directories may contain capabilities for other directories, hierarchical file systems can be built easily, as well as more general structures. A directory entry may contain either a single capability or a set of capabilities, to allow a file name to map onto a set of replicated files. When the user looks up a name in a directory, the entire set of capabilities is returned, to provide high availability. These replicas may be on different file servers, potentially far apart (the directory server has no idea about what kind of objects it has capabilities for or where they are located). Operations are provided for managing replicated files in consistent way.

C. Compilers

Amoeba comes standard with compilers for ANSI standard C, Pascal, Modula 2, BASIC, and Fortran 77. Each of these comes with appropriate libraries. Amoeba also comes with a collection of third-party software, including the GNU C compiler.

D. Parallel Programming

A new language called Orca has been developed. It is for parallel programming. Orca allows programmers to create user-defined data types which processes on different machines can share in a controlled way, in effect simulating an object-based distributed shared memory over a LAN. Operations on each object are performed in such a way as to provide the illusion of there being only a single copy, shared by all machines. The Orca run-time system uses the Amoeba IPC facilities to make sharing of software objects over the network highly efficient.

E. TCP/IP

Although the basic communication mechanism in Amoeba is the Amoeba FLIP protocol, a special server is provided to allow TCP/IP communication, through RPCs to the TCP/IP server. In this way, machines can be accessed through the Internet.

F. XWindows

Amoeba's user interface is the industry standard X Window System (X11R6). For X servers running on workstations, a special version of X is available that uses the Amoeba RPC for high-performance communication. When hard-wired X terminals are used, these can be interfaced using the TCP/IP server.

VII. NONTECHNICAL ASPECTS OF AMOEBA

A. Source Code Availability

All academic Amoeba distributions contain the entire source code. Binaries for the supported machines are also included.

B. Amoeba is unencumbered by AT&T licensing

Amoeba was written from scratch. Although it provides a partial POSIX emulation, it contains no AT&T code whatsoever. Furthermore, the utility programs it comes with have either been written from scratch or obtained from third parties under favorable conditions. Although customers are required to agree to our license, no additional licensing is needed for Amoeba.

C. Machines on which amoeba runs

Amoeba currently runs on the following architectures:

- Sun 4c and MicroSPARC SPARCstation
- Intel 386/486/Pentium/Pentium Pro (IBM AT bus, PCI bus)
- 68030 VME-bus boards (Force CPU-30)

Sun 3/60 & Sun 3/50 workstation

VIII. CONCLUSION

Amoeba is a modern distributed operating system that is designed for an environment consisting of multiple computers.

REFERENCES

[1] Kaashoek, M.F., Renesse, R. van, Staveren, H. van, and Tanenbaum,A.S.:"FLIP:an Internetwork Protocol for Supporting Distributed Systems," ACM Trans. on Computer Systems vol 11, pp. 73-106, Feb. 1993.

[2] Andrew S. Tanenbaum & Gregory J. Sharp Vrije Universiteit "Amoeba Distributed Operating System"