Originally published at pp. 594-97 of Coulouris, Dolllimore and Kindberg, Distributed Systems, Edition 2, 1994.

A comparison of Mach, Amoeba and Chorus

Amoeba is a complete and novel distributed operating system constructed as a collection of user-level servers supported by the microkernel. Mach and Chorus are primarily microkernel designs geared towards the emulation of existing operating systems, notably UNIX, in a distributed system.

Mach, Chorus and Amoeba have many common general goals, including the support of network transparency, encapsulated resource management and user-level servers. In Mach and Chorus, some objects are managed by the kernel and others by user-level servers, whereas all Amoeba objects are managed outside the kernel. Chorus allows user-level servers to be loaded dynamically in the kernel address space.

Mach, Chorus and Amoeba all provide separate abstractions of processes and threads. Mach and Chorus can take advantage of a multiprocessor.

The Amoeba kernel interface is very simple, because of its simple communication model and lack of support for virtual memory. The Mach and Chorus kernels offer many more calls due to their more complex communication models and the desire to emulate UNIX.

Naming and protecting resources ◊ Resources are named and protected by capabilities in Amoeba and Chorus and by ports in Mach. Resources identified by capabilities in Amoeba and in Chorus are accessed by sending a message to the appropriate server port and the server accesses the particular resource identified in the capability. In Mach, servers generally manage many ports, one for every resource. Resources are accessed by sending messages to the corresponding ports.

Capabilities alone are not suitable for implementing the sort of identity-based access control required in UNIX file systems. The Chorus kernel provides protection identifiers to enable user-level services to authenticate the actor that sent a message and the port used by that actor.

Mach's port rights are capabilities that confer send or receive rights on the process that possesses them. However, unlike Amoeba's capabilities and Chorus's port identifiers, which can be freely constructed and manipulated at user-level, Mach's port rights are stored inside the kernel and protected by it, allowing efficient representations and rapid access. But the Mach kernel has the additional expense of processing port rights in messages.

For local communication, the Mach approach eliminates the need for random number generators and one-way functions, which are associated with Amoeba capabilities. However, in a secure environment, the Mach network servers must encrypt port rights transmitted in messages.

Amoeba capabilities and Chorus capabilities can persist beyond the execution of any process that uses them. An Amoeba capability for a persistent resource such as a file can be stored in a directory. Each component of the capability, in particular the port identifier, remains valid as long as the file exists and the check field has not been changed at the server. By contrast, Mach capabilities for send rights are volatile. To access a resource, a Mach client requires a higher-level, persistent identifier to obtain the current identifier of the appropriate send rights; this higher-level identifier must be resolved by the service concerned before access can be obtained.

There does not seem to be a clear winner between Mach's scheme of kernel-managed port capability transfers, and the Chorus and Amoeba scheme of user-controlled capability transfers. In Amoeba, processes are obliged to generate port identifiers themselves, and then test for their

uniqueness. The Chorus port naming scheme improves upon this since the kernel generates UIs for port identifiers, thus avoiding clashes.

Both Amoeba and Chorus allow for groups of servers to manage resources. In Amoeba the processes form groups, but in Chorus processes effectively become members of groups by making their ports join port groups.

Interprocess communication ◊ The Mach, Chorus and Amoeba kernels all provide a synchronous request-reply protocol. Chorus and Mach also provide for asynchronous message passing. Mach packages all forms of message passing in a single system call, whereas Chorus provides alternative calls.

Amoeba and Chorus provide for group communication. In the case of Amoeba this is a reliable, totally ordered multicast to be used by the members of a process group. Chorus provides an unreliable multicast to all the members of a group of ports or to selected members. Chorus services can be reconfigured by servers adding or removing their ports from a group.

Messages ◊ Amoeba messages consist of a fixed-size header and an optional out-of-line block of data of variable size. But Mach is more flexible in that it allows multiple out-of-line blocks in a single message. Mach and Chorus employ their virtual memory management techniques to the passing of large messages between processes in the same computer.

The contents of Mach messages are typed – enabling port rights to be transmitted. Messages in Chorus and Amoeba are just contiguous sequences of bytes.

Network communication ◊ Communication between processes in different computers is performed in Mach and Chorus by user-level network servers running at every computer. The network servers are also responsible for locating ports.

The Amoeba kernel supports network communication directly, and is highly tuned to achieve rapid request-reply interactions over a LAN. The Amoeba designers considered that the extra context switching costs that would be incurred through use of a separate, user-level network manager process would be prohibitive,

Mach, Chorus and Amoeba use similar schemes for locating ports. Location hints are used first but if those fail they resort to broadcasting.

The use of user-level network servers should allow for a variety of protocols. However, Mach's network servers primarily use TCP/IP as the transport protocol. Chorus also sticks to international standards, providing Internet and OSI protocols. However, this is not necessarily suitable on LANs when request-reply interactions predominate.

The Amoeba kernel supports an RPC protocol based upon the FLIP datagram protocol, which is optimized for performance on small sets of connected LANs.

A figure of 11 milliseconds for a null RPC in Mach on a Sun-3/60 is quoted by Peterson *et al*. [1990]; for a DECStation 5000/200 the figure is 6.3 milliseconds [Orman *et al*. 1993]. The poorness of these figures compared to, say, Amoeba's of 1.4 milliseconds on a 68020-based computer is in part due to the transport protocol used. It is not totally due to this factor: a conventional UNIX implementation on a Sun-3/75 takes 6.1 milliseconds using TCP [Peterson *et al*. 1990]. (All figures are for a 10 megabits-per-second Ethernet.) Recently, Mach IPC has been re-implemented using the x-kernel, with the transport protocols placed in the kernel to improve performance. Orman *et al*. [1993] quote an improved null RPC figure of 4.8 milliseconds.

Memory management ◊ Amoeba has very simple memory management scheme without virtual memory. In contrast Mach and Chorus provide similar very powerful and flexible virtual memory management schemes allowing a variety of different ways of sharing between processes, including copy-on-write.

Mach and Chorus both make use of external pagers and local caches, which allow virtual memory to be shared between processes, even when they run in different computers.

UNIX emulation ◊ Amoeba does not provide binary compatibility with UNIX. Chorus and Mach both provide emulation of UNIX as subsystems.

References

[Peterson *et al*. 1990] Peterson, L., Hutchinson, N., O'Malley, S. and Rao, H. (1990). The x-kernel: A Platform for Accessing Internet Resources. *IEEE Computer*, vol. 23, no. 5, pp. 23–33.