Unit 15:
Experimental Microkernel Systems

15.3. Comparison of Amoeba, Mach, and Chorus

Philosophy – Computer vs. Cluster

- Amoeba:
  - Based on processor pool model
  - User logs into the system as a whole
  - OS decides where to run commands based on load
  - Optimized for remote case (fast RPC)
- Mach and Chorus:
  - User logs into a specific machine
  - No attempt to spread each user‘s work over machines
  - Each user has a home machine - but Mach was ported to the Intel Paragon multiprocessor, consisting of a pool of processors
  - Optimized for local case (copy-on-write in Mach memory management)

Philosophy - Microkernel

- Amoeba:
  - Perfection is not achieved when there is nothing left to add, but when there is nothing left to take away (Atoine de St. Exupéry)
  - Minimal kernel, most code in user-space servers
- Mach:
  - Provide enough kernel functionality to handle wide range of apps.
  - 4.2BSD UNIX compatibility
  - Large kernel, five times more system calls than Amoeba
- Chorus
  - Smaller than Mach kernel
  - Still more system calls than 4.2BSD UNIX

Objects and Capabilities

- Amoeba:
  - Objects are the central concept
  - Few are built-in, most are user defined (e.g. Files)
  - About a dozen generic operations on objects
  - Capabilities managed in user-space; for system/user-defined objects
- Mach:
  - OS objects:
  - Capabilities only for ports; not for processes/other system objects
- Chorus:
  - Built-in OS objects: threads, processes, ports, memory segments
  - Subsystems may define new protected objects
  - Capabilities for all objects; no encryption of right fields

Processes and Threads

- All systems support processes with multiple threads
- Amoeba and Chorus:
  - Thread synchronization by mutexes and semaphores
  - No primitives for assigning threads to processors
  - Automatic load balancing in processor pools (Amoeba)
- Mach:
  - Thread synchronization by mutexes and condition variables
  - Programmer may manage thread-to-processor assignment
  - Load balancing only on multiprocessor systems

Memory Model

- Amoeba:
  - Variable-length segments, no paging
  - Segments are controlled by capabilities
  - Shared objects of any size (impl. based on reliable broadcast protocol)
- Mach:
  - Memory objects, fixed-size pages
  - Page fault handling by external user-space memory managers (OS supplies default memory manager)
  - Copy-on-write page sharing (optimization for multiprocessor systems)
- Chorus:
  - Memory objects (regions)
  - Demand paging under control of an external pager (Mapper)

Communication

- Amoeba:
  - RPC (simple interface) and group communication
  - Put-ports represent service addresses
  - Ports are cryptographically protected (via one-way functions)
- Mach:
  - RPC communication, mapped onto memory manag. for local ops.
  - Remote communication handled by user-space server (netmsgserver)
  - No group communication or reliable broadcasting as kernel primitives
- Chorus:
  - Messages are directed to ports; similar to Mach
  - RPC or asynchronous communication
  - All communication implemented inside the kernel

Servers

- Amoeba:
  - Variety of servers for specific functions
  - File/directory management, object replication, load balancing
  - All servers are based on objects and capabilities
  - UNIX emulation provided at source code level
- Mach:
  - Single server runs BSD UNIX as an application program
  - 100 percent binary-compatible emulation
- Chorus:
  - Full binary compatibility with System V UNIX
  - Emulation implemented by collection of processes (like Amoeba)
  - Native servers designed from scratch; distributed computing in mind