Mach : A New Kernel Foundation For UNIX Development

Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian and Michael Young

Carnegie Mellon University

Overview

- Developed at Carnegie Mellon
- Replacement kernel for UNIX/BSD
- Derived from Accent
- A new foundation on which a UNIX like systems can easily be developed
- Existing UNIX applications should run without modification on Mach

Architecture

- Microkernel architecture so the kernel should provide only a few basic features

- Majority of features are provided by user level service processes

- User processes communicate using IPC mechanisms like message passing

- BSD functionality packaged into a single process called the POE server which ran in the kernel, later to be moved to user space

Features

- Multiprocessor support
  - Tightly and loosely coupled multiprocessors
- New virtual memory design
  - Allows allocation and deallocation of memory
  - Specify protection boundaries and inheritance of memory regions
- IPC facility
  - Works transparently across local systems as well as network boundaries
- System support facilities like a kernel debugger and remote file system

Abstractions

- Task
- Thread
- Port
- Message

- Operations can only be performed on messages

- Other operations are performed by sending a message to the relevant port

Tasks

- Execution environment and basic unit of resource allocation
- Contains virtual address space and access restrictions
- Can run a number of threads
- Similar to the concept of a process in UNIX

Threads

- Unit of execution
- Threads within the same task share memory resources
- Lower overhead than creating a new task/process
- Parallelism in Mach
  - Multiple threads within a single task
  - Shared memory across different tasks
  - Message passing between various tasks

Virtual Memory Management

- Manages virtual memory during forks

- Memory pages have inheritance values – shared, copy or none

- Shared pages are shared for reading and writing between parent and child process

- Copy pages are copied into the child process, but copy on write is used

- None pages are not shared

Virtual Memory Implementation

- Address maps – doubly linked lists containing memory linked to a task

- Shared maps – describe memory shared between tasks

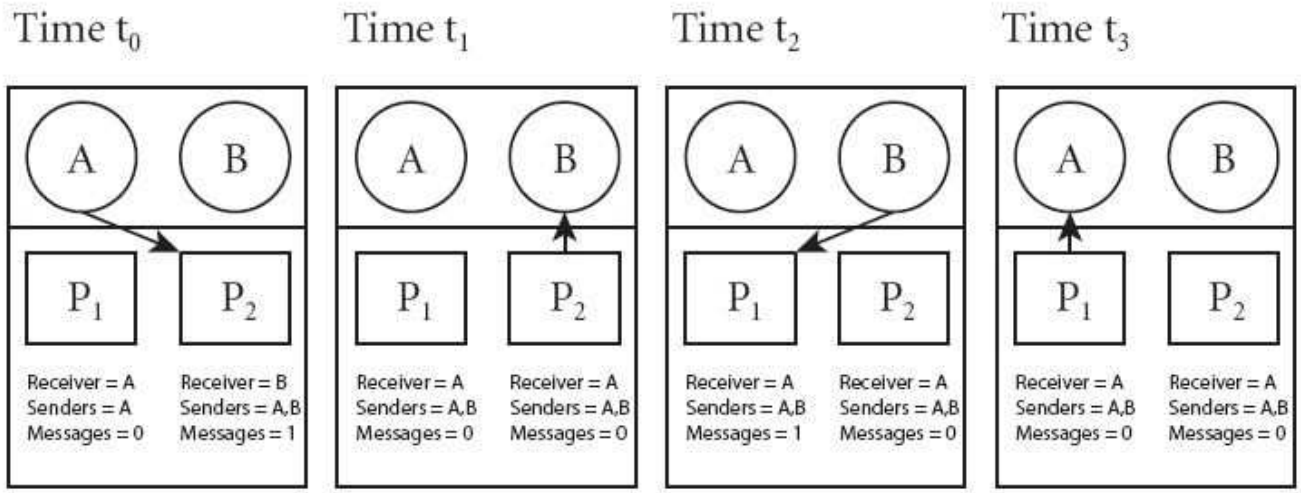
- VM objects – specifies resident pages and location of non-resident pages. Shadow VMs are used for copy-on-write pages

- Page structures – attributes for physical pages

Interprocess Communication

- Uses ports and messages
- Port – finite length queue of messages sent to a task
- Processes expose ports onto which other processes can send them messages
- Multiple tasks can write to a single port, but only a single task can read and process the messages on it
- Permissions can be set to decide if a task can write to a particular port

Message Sending



- A is sending a message to B, so sends to the port P_2 exposed by B
- B reads the message from the port, processes it and replies by writing to port P_1, exposed by A
- 4 switches between kernel and user mode

Network Communication

- Remote calls handled transparently by user level tasks, so sender is unaware if the eventual port is local or remote

- User level task exposes a local port to which other tasks can send messages to

- Forwards the messages across the network to a task running on a remote system and listens for the reply

- Reply is sent back to the task that had originally sent the message to the local port

Conclusion and Legacy

- One of the first functional micro-kernel architecture
- New thread model which allowed a single task or process to have multiple threads
- Plagued by performance problems that plagued wide scale adoption
- Mach kernel, along with BSD code, forms the basis for the hybrid kernel (XNU/Darwin) used in Mac OS X

Onto the Discussion

Performance

• Why is Mach slow? What are the sources of the overhead which cause it to be 50-75% slower than BSD/UNIX?

• Are the performance issues specific to Mach, or to any IPC based microkernel which makes a greater number of context switches between user and kernel mode?

• Are microkernels viable given the performance issues? Are there any workarounds for these performance issues?

• Is there always to be a trade-off between performance (having everything in the kernel) and fault tolerance (having lots of drivers in user space systems)?

IPC

- How are ports exposed by tasks located?
- If multiple tasks can write to the port of a single task, how are they scheduled?
- How can priorities be implemented on message queues?
- How can Mach control the privileges/capabilities necessary for different user level modules to execute?

Current State of Affairs

- Is Mach, or any improved derivative of Mach, still in use in any modern OS?

- Has Mach ever been implemented by removing the UNIX compatibility layer (POE server) from the kernel and moving it into user space? Have there been any practical applications of such an implementation?