MACH: A New Kernel Foundation for UNIX Development

M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young

Presenter: Wei-Lwun Lu

Introduction

- The MACH system
  - Developed by Carnegie Mellon University from 1985-1994
  - Derived from Accent
  - A replacement of UNIX 4.3 BSD kernel
- MACH features:
  - Support for multi-processors
  - New virtual memory design
  - Capability-based inter-process communication facility
  - System support facilities (e.g., a kernel debugger)

MACH Concept

- Microkernel (from MACH 3)

  kernel only provides basic features, services are run as user-mode services
- Mach has four basic abstractions:
  1. Task
  2. Thread
  3. Port
  4. Message
- MACH advantages
  - Better extensibility
  - Better performance (based on experiment on Accent)

Tasks and Threads

- **Tasks**
  - Basic unit of resource allocation
  - Include a virtual address space and access to system resources
- **Threads**
  - Basic unit of CPU utilization
  - All threads within a task share system resources
- Mach was the first one that implemented the task/thread system
- Application parallelism is achieved in any of three ways:
  1. Multi-threading (within task)
  2. Share memory (between tasks)
  3. Message passing (between tasks)

Virtual Memory Management

- MACH has a default memory manager
- The Mach virtual memory design allows tasks to
  1. Allocate memory
  2. De-allocate memory
  3. Set protections on regions of virtual memory
  4. Specify the inheritance of regions of virtual memory (shared, copy, none)
- Example: fork operation in UNIX
- Example: user-level memory manager (handle page faults and page-out in user-mode)

Virtual Memory Implementation

- Virtual memory implementation is split into
  - Machine dependent sections (e.g., VAX Page Tables)
  - Machine independent sections
    1. **Address maps**

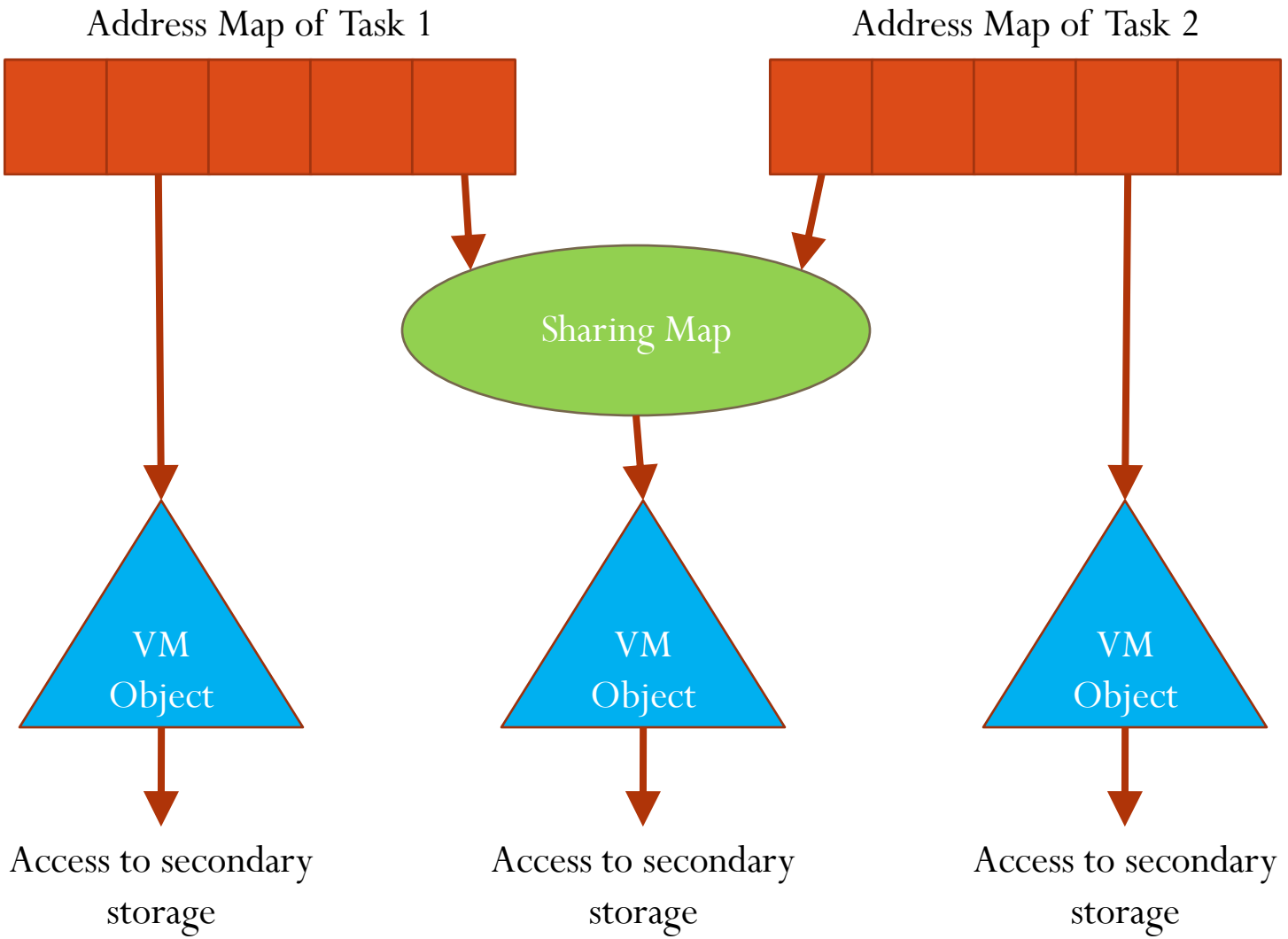
       A doubly linked list of map entries associated with each task
    2. **Share maps**

       Special address maps that describe regions shared between tasks
    3. **VM objects**

       Manage secondary storage. Allows easy experimentation with new memory-manipulation algorithms.
    4. **Page structures**

       Specify the current attributes for physical pages

Virtual Memory Implementation



Interprocess Communication

- IPC in Mach is defined in terms of ports and messages
  - Advantages: provide location independence, security, and data type tagging
- **Port**
  - A finite length queue of messages sent by a task
  - May have any number of senders but only one receiver
- **Message**
  - Message consists of a fixed length header and a variable size collection of typed data object
  - Synchronously or asynchronously
  - Copy-on-write for better efficiency

Interprocess Communication

- Implementation
  - Rely on a language called Matchmaker
  - Compile IPC into remote procedure call (RPC) stub which uses MACH message passing as its transport facility
  - Matchmaker performs runtime type-checking (overhead)
- Network communication and security
  - The Mach sends IPC to a network server
  - Network server
    - Local representatives for tasks on remote nodes
    - Handle communication security
    - Maintain a mapping between network port to their corresponding local ports

System Support Facilities

- Kernel Debugger
  - Build-in kernel debugger (kdb) based on adb
  - Functionalities: breakpoints, single instruction step, stack tracking, symbol table translation, etc.
- Transparent Remote File system
  - A small set of kernel hooks redirects remote file operations to remote servers transparently

Conclusion

- The MACH system
  - Designed as a replacement for the UNIX 4.3 BSD kernel
  - Smaller kernel, move many kernel services to user-level
  - New task/thread design, virtual memory design, and interprocess communication
  - Goal:

    "kernelize" UNIX to a substantially less complex and more easily modifiable basic operating system

Questions

- Performance concerns:
  - The main gain of Mach seems to be that it reduces the "responsibility" of the kernel, and most things are done in user-space with communication via IPC. But doesn't this use of IPC impose a dramatic speed-hit?
  - How fast is the Mach system? It seems that all this flexibility and simplicity through use of simple primitives will come at a large overhead. The researchers are optimistic, but I'm still sceptical.

Questions

- Why didn't MACH replace UNIX?
  - Mach claims to have similar (or better) performance than UNIX (4.3BSD), run UNIX code, but have the added advantage of being able to be run on multiple processors (both locally and remotely). Why didn't Mach replace UNIX (and other *nix operating systems)?
  - Currently Mach approach, which can generally be thought as an IPC based operating system, is considered as inherently flawed (according to wikipedia). What can be the main reason for Mach not to be accepted as a successful operating system?

Questions

- Kernel debugger's effectiveness:
  - The paper claims that Mach has a built-in debugger, and it makes sense for the user to be able to enter a "debug mode", etc. But how (if at all) is this debugger able to handle a crash?
  - It is my understanding that debugging at the kernel level is still a difficult task. The authors mention their implementation of the kernel debugger (kdb), which seems to have all the functionality of a user-level program debugger. How come such debuggers haven't gained popularity?

Questions

- How to find ports?
  - Since Mach's "ports" can appear and disappear dynamically, how do software developers conveniently find out which ports and their corresponding services exist? It would appear that Mach's flexibility would hinge upon convenient mechanisms given that Mach's object-based structure is fundamentally dependant on the premise of a "port".

THANKS