

# The Mach System



- Appendix B of Operating Systems Concepts, Sixth Edition by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne
- Presented by Joseph Jess

# History

- Older kernel design can do it all, but not as safely as Mach
- Mach began mostly as BSD UNIX 4.2
  - As work was finished on modules for Mach they were put in place of the BSD modules they were designed after,
    - When BSD 4.3 came out Mach was also updated
- Most kernel code was moved to user-level as servers to reduce kernel usage, which began a new idea in systems design; the micro-kernel
  - Leaving the Mach primitives in the kernel allows systems to be built on top of Mach

# Monolithic versus Micro

- What is a monolithic kernel?

# Monolithic versus Micro

- What is a monolithic kernel?
  - A monolithic kernel has most of the functionality of a computer system built into the kernel, and while this may create a fairly stable foundation to run user-level programs on, it is not very safe because one small bug could cause the system to corrupt or halt.

# Monolithic versus Micro

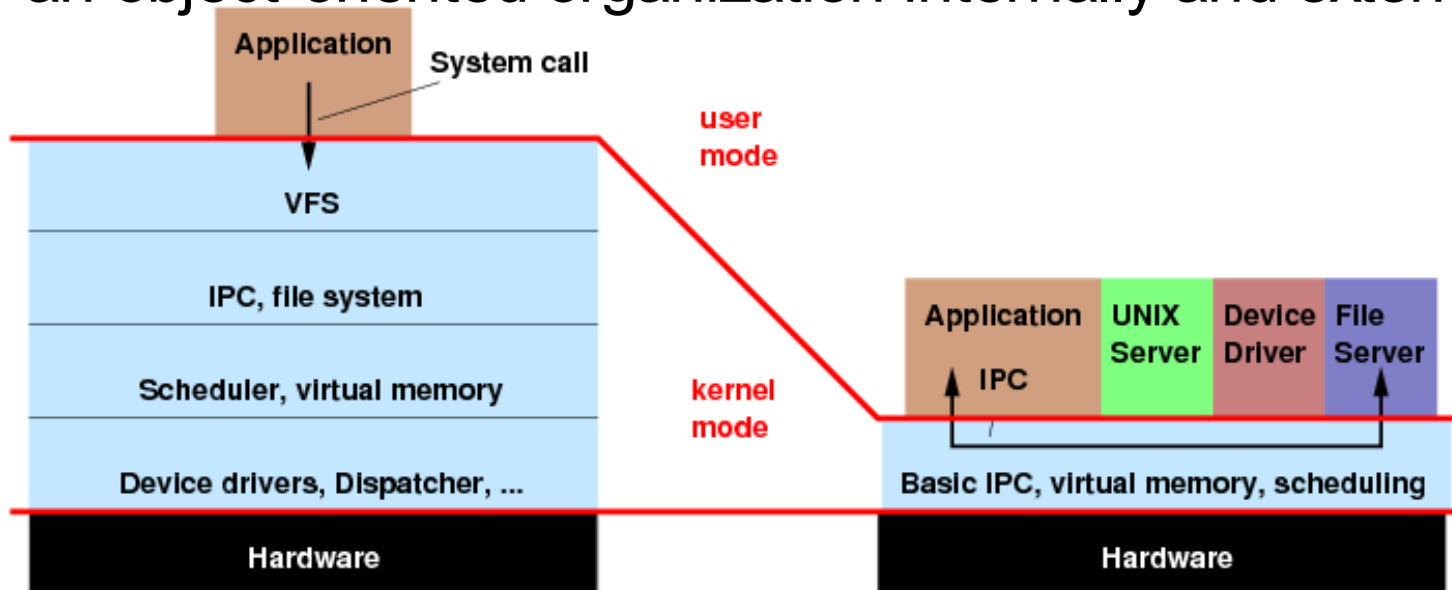
- What is a monolithic kernel?
  - A monolithic kernel has most of the functionality of a computer system built into the kernel, and while this may create a fairly stable foundation to run user-level programs on, it is not very safe because one small bug could cause the system to corrupt or halt.
- What about a micro-kernel, what makes it different from its larger relative?

# Monolithic versus Micro

- What is a monolithic kernel?
  - A monolithic kernel has most of the functionality of a computer system built into the kernel, and while this may create a fairly stable foundation to run user-level programs on, it is not very safe because one small bug could cause the system to corrupt or halt.
- What about a micro-kernel, what makes it different from its larger relative?
  - A micro-kernel has only basic tools to run an operating system, while the code that actually implements higher functionality is outside the kernel in some form such as servers. While this may allow greater safety, it also comes at a price due to communication.

# Concept to Systems design

- Micro-kernel design has many benefits
  - Systems can run on top of the micro-kernel
  - User-level libraries for increased capabilities and efficiency
  - Distributed operation provides network transparency and an object-oriented organization internally and externally



# Unix-like benefits

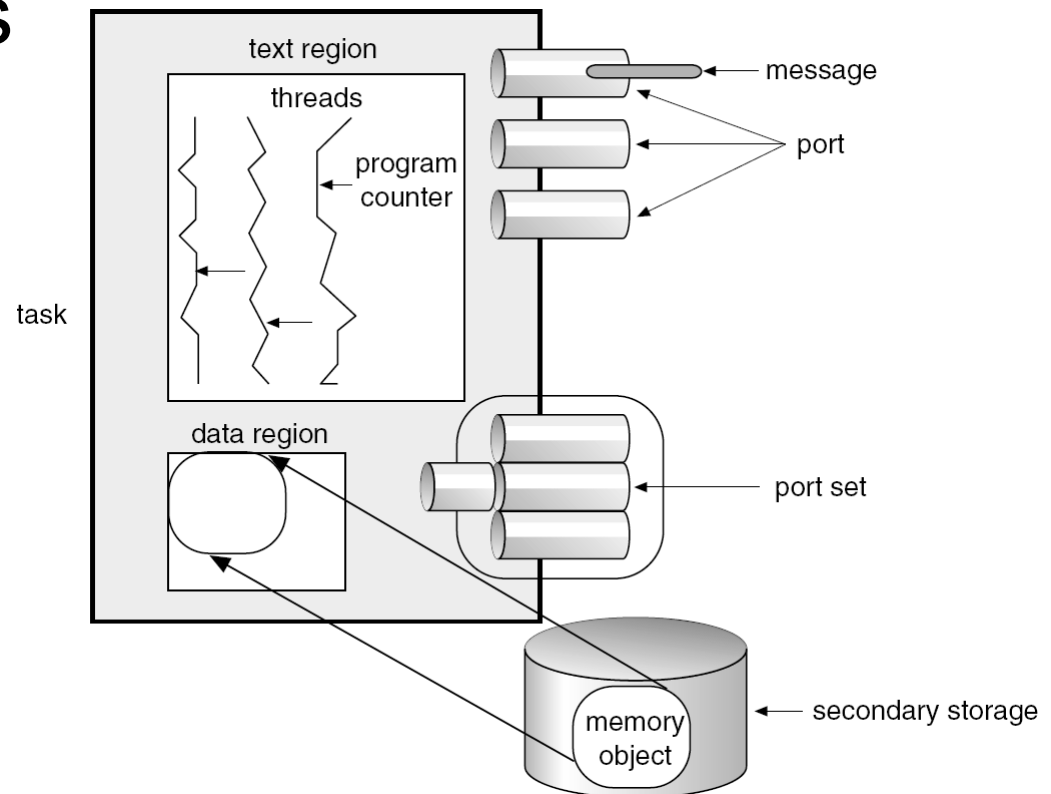
- Being UNIX-like provided many benefits
  - Simple programmer interface with good primitives and consistent interfaces to system facilities
  - Easy portability to a wide range of machines
  - Extensive libraries of utilities and applications
  - Powerful pipes for many purposes

# BSD drawbacks

- Being created from BSD has some drawbacks
  - Redundant features in the kernel repository makes management and modification more difficult
  - Original design goals made multiprocessing difficult to achieve; e.g. Having no provisions for locking code or data that others may be using
  - Too many fundamental abstractions makes accomplishing a task more difficult

# System Components

- Basic component abstractions make Mach's micro-kernel a diverse and powerful system without becoming large quickly like previous operating systems



# System Components (exec.)

- Some abstractions are for execution purposes:
  - A task is an execution environment that is used as the basic unit of resource allocation.
    - A task may contain multiple threads, and is composed of a virtual address space with protected access to system resources
  - A thread is the basic unit of execution, and must run within a task's protection.
    - Note that there is no notion of a process, instead a process would be a single threaded task

# System Components (msgm.)

- Other abstractions are for managing messages
  - A port is the basic reference to objects, and is a kernel-protected communication channel.
    - Ports are protected by the kernel as *port rights*, that is a task must have a port right to send a message to a port.
  - A port set is a group of ports sharing a common message queue.
    - A thread can receive messages from a port set, thus servicing multiple ports. Each message contains the port it was received from, thus identification of the object referred to by the message

# System Components (other)

- And still others have different uses
  - A message is the basic method of communication between threads. it is a typed collection of data objects.
    - Each message may contain actual data or out of line data references. Port rights are also passed via message.
  - A memory object is a source of memory; tasks may access it by mapping portions (or the entire object) into their address space.
    - Memory objects may be managed by user-mode *external memory managers*; an example of such an occurrence would be a file managed by a file server.

# Object use

- What do you think these objects and ideas could be used for when we want efficient cross domain communication?

# RPC of course

- An implementation of RPC is built into Mach, providing great benefits using the efficient objects available in Mach
  - RPC is used to synchronize exception-causing thread and handler thread, allowing exceptions to be simply defined and able to be used by user-level code
  - RPC allows Mach to support BSD-style signals, providing interrupts and exceptions for BSD program support
    - for programs that do not support this there is a server to receive and translate exceptions into the appropriate signals

# Memory and IPC

- Oddly, memory objects are represented by ports, which means certain qualities are present
  - IPC is used to request operations upon the object. This allows remote and transparent access to these objects
  - Flexibility in memory management in user programs
  - Generality allowing virtual copies in tightly or loosely coupled computers
  - Improvement over basic UNIX message passing
  - Easy task migration because tasks are self contained and port communication is still possible even when moved to another computer thanks to the NetMsgServer

# Process Management

- A task can be compared to a UNIX process
  - Fork creates a new task with a single thread, and will have a duplicate address space, according to the *inheritance attributes*, of the parent
  - Threads are extremely useful in server applications, since one task can have threads manage the multiple requests
    - These threads can be spread over many processors, and even if a page fault occurs the remaining threads may continue executing

# Process Management Cont.

- A thread or task may be in one of two states
  - Running; being executed or waiting for allocation
  - Suspended; waiting to be returned to the running state
  - A task that has been suspended has all threads in it suspended
  - A suspend count is kept for each thread, where an equal amount of resume calls occur is the thread resumed

# Threads Basics

- Threads are made by C-Threads which allows
  - Creation
  - Destruction
  - *Wait*
  - *Yield*

# Threads Advanced

- Threads sharing data can be managed using built in Mach functionality
  - Mutual exclusion is achieved through spinlocks
    - *mutex\_alloc* creates a mutex variable
    - *mutex\_free* deallocates a mutex variable
    - *mutex\_lock* locks a mutex variable (does not prevent deadlock)
    - *mutex\_unlock* unlocks a mutex variable, much like a typical *signal*
  - And general synchronization with condition variables
    - *condition\_alloc* creates a condition variable
    - *condition\_free* deallocates a condition variable
    - *condition\_wait* unlocks the mutex variable and blocks until *condition\_signal* is executed on the condition variable, and then the mutex is locked and the thread continues

# CPU Scheduling

- Multiprocessor scheduling is more complex than standard UNIX-based process scheduling.
- Mach uses a simple policy to prioritize threads
  - Each thread has a priority number ranging from 0 to 127 based on the exponential average of its usage of the CPU
    - More recent or longer use of the processor means a lower priority
    - This priority is used to group threads into one of 32 global queues
  - There are also local queues to each processor for device drivers and other objects connected to a single processor
  - This means that each processor is responsible for finding threads to run, not a central dispatcher

# Some Port Details

- A port is a protected, bounded queue within the kernel that the object resides. There are system calls that can provide uses to tasks
  - If the queue is full, a sender may abort and wait or have the kernel deliver the message for it
  - Ports may be allocated to tasks, where the caller gains rights to the new port
  - Ports may be deallocated and all holders of send-rights may potentially be notified
  - Port status may be retrieved
  - Port backups may be created to receive messages in the case that a task containing receive rights deallocates the port being backed up

# Some Message Details

- A message consists of
  - A header with port name, reply port, and length
  - Some In-line data which may be copied directly into the message
  - Some Out-of-line data which is referenced via pointer
  - Any data section may be simple type, port rights, or pointers; each section is typed so that inter-machine communication will not affect the data being transferred
  - Pointers in message must be processed by the kernel because addresses would be invalid in any address-space other than the sender's address-space

# NetMsgServer

- The *Network Message Server* is a user-level capability-based service that forwards messages between hosts
  - To communicate with a task you would have have a port of a task, so there is a primitive service that allows tasks to register ports for lookup by tasks on any other computer in the network
  - Given that a task has the port rights to another task all a task has to do is send off to that port and the rest is transparent

# NetMessageServer

- Type information from sender's message is translated so that receiver understands the data correctly
- The NetMsgServer manages adding, looking up, and removing ports from the NetMsgServer's name service

# Memory Objects

- Memory objects in Mach are very important, they have many beneficial qualities such as
  - Used to manage secondary storage
  - Represent files, pipes, or any other data that can be mapped into virtual memory for reading and writing
  - May be backed by user-level memory managers
  - Memory objects are like any other object in the system
    - They have ports, can hold data, and can be mapped directly into a task's address space

# User-Level Memory Managers

- Being as memory can be organized so easily, it is possible to use user-level memory managers to manage portions of memory
  - Allows less overhead in the kernel for managing data
  - But it is up to the memory manager to back changed pages into storage after an object is destroyed.
  - No assumptions are made about content or importance of memory objects
- Not all situations can be handled by user-level memory management, so there is a default memory manager built into Mach

# Shared Memory

- Sharing data is often important in a system so Mach has ways of sharing data
  - Intra-task requires nothing special since tasks each have their own memory that threads from that task can access
  - Parent tasks that fork may declare through inheritance rules which regions are inherited by its children, and which are read-writable
  - External memory managers may hold a section of memory that can be used by nonlocal tasks to a machine

# Programmer Interface

- Programmers can work at many levels in Mach
  - System call level
  - *C Threads* package allows threading at the user-level
- Mach is a multi-server model
  - Features available through user-level libraries
  - Features available through servers
  - Many Features removed from the kernel
  - several operating systems could be run on top of Mach simultaneously if desired

# Programmer Interface Cont.

- Mach primitives
  - are flexible, they can be used for many things including running several operating systems already
  - make programming repetitive, being as you would have to program message calls in each program you code
    - though there is a way to reduce this with an interface generator called *MIG* which codes stubs for the interfaces you provide to *MIG* for message passing code to be generated for the programmer

# Summary

- Mach is designed with many of the newest features of its time, and was actually used in many operating systems that came afterward; with some alterations.
- To say that this system is formidable would be an understatement, this system helped define the direction of future operating systems.