Greenlee 560 Summary 1

Elliot Greenlee

January 26, 2017

1 Introduction

In recent years there have been many innovations in operating-system technology. This paper gives details of the Mach operating system, a new, functional system designed to incorporate these new improvements into a technically advanced OS.

1.1 History of Mach

Mach's oldest philosophical ancestor is the Accent operating system, which was developed at Carnegie Mellon University. Mach was originally developed inside of the 4.2 BSD kernel. First, BSD kernel components were replaced by Mach components as they were finished. By Mach 3, BSD and Unix specific code was removed from the kernel to run in user-mode, leaving only basic Mach features.

2 Problems

What problems does this paper try to solve? Why are such problems important? Overall, the Mach designers saw an issue with other contemporary operating systems.

2.1 Current Operating Systems

Current operating systems often do not tackle the wide scope of all new improvements to OS technology. Additionally, operating systems today are often targeted at a specific architecture, and are not designed to handle diverse architectures.

It is important for new operating systems to incorporate the results of research in the area. As hardware growth improvements slow, more performance optimization needs to come from software. At the core of most software will be the operating system, providing organizational efficiency and features.

2.2 BSD/Unix

The modular nature of BSD, coming from the multitude of abstractions at its core, has caused the kernel to hold many redundant and competing ways to complete the same goals. Additionally, BSD was not designed with multiprocessing, distributed systems, and shared program libraries in mind. This has resulted in problems when trying to perform these tasks, including security vulnerabilities with shared users.

Although BSD provided much of the starting place for MACH, it is important that the designers recognized its limitations. Multiprocessing must be addressed, and solutions must be provided as the current trends in technology proceed. Most computers have multiple processors, and many server and virtual machine systems are constantly being accessed my multiple users.

3 Assumptions

What are assumptions made by this paper? Are they verifiable? Are there logical holes in such assumptions?

Overall, the Mach designers made certain design assumptions about the desires of their users, and best practices for the operating system. These assumptions have so far been shown to be correct.

3.1 BSD Principles

The developers begin with the assumption that BSD is a good place to start. It has a simple user interface, a uniform interface, and a well chosen set of primitives. Easy portability mixed with an extensive library of utilities that are easily combined via pipes provided the fundamentals to begin.

Although there is not a formal logical way to derive design principles, BSD is verified through its popularity and by much testing from different environments. However, BSD caused problems for the team as they developed Mach. Many of these derived from the uni-processor focused nature of BSD and Unix. This led to unnatural forced implementations in BSD, revealing that perhaps starting from scratch would have been the most clean approach. Signals were designed for only single-threaded programs, and in Mach all tasks are threaded; tasks are scheduled by the operating system at the thread level. However, because the signal system must work to be compatible with BSD, they could not be removed.

3.2 Threads

The designers chose thread level execution for tasks, hoping for increased efficiency for multiprocessing. Tasks can have threads on different parallel processors. At the user level, threads allow paging to only influence the efficiency of a single thread. This efficiency has been verified through testing, and in its current version, the Mach system is comparable with other UNIX versions.

Despite the benefits, threads also have an associated cost. Memory usage is increased in the kernel to store the supporting data structures, and efficiency is reduced through the effects of the complex scheduling algorithms needed.

3.3 Research

The designers attempted to create an operating system which made research easier, by designing a kernel even simpler than 4.3 BSD. Researches can use user-level code to implement new features, rather than tailoring an operating system. Additionally, these features can be implemented at a higher level such as C, which provides access to the primitives. Again, the success of this goal is verifiable through usage. The Open Software Foundation (OSF) announced in 1989 that it would use Mach 2.5 as the basis for its new operating system, OSF/1, and so Mach became, and has remained, much more popular.

Despite their attempted improvements, Mach still has problems for researchers. Mach's flexibility is derived from the user of primitives, by these primitives make programming more repetitive. In order to send and receive messages, which is required for most tasks, large portions of code are needed.

4 Solutions

What are the major solutions of this paper? Do you think the solutions in this paper will work for the problem? Do you think the paper evaluates the solutions in a convincing manner? List two limitations of the solutions proposed in this paper, and outline your method to fix them.

Overall, the solutions provided by Mach are well thought out and have been tested over many years. This description evaluates the positives and negatives of solutions, including possible limitations and the steps Mach has taken to fix them.

4.1 Small Kernel

The designers kept the the Kernel very small, but easily extensible. In order to reduce the size, low-level system calls are provided to produce more complex functions. Operating-system emulation occurs at the user level, in a similar way to virtual-machine systems. Additionally, member objects are independent of the kernel, which no assumptions made about the contents or importance.

The test of time and popularity has shown that this fundamental design goal has been successful. This paper provides the positives and negatives of this solution. BSD also had a very small set of core abstractions, which caused redundancy issues later in its development. Additionally, functionality must be developed and added to the kernel, rather than the kernel containing basic features to which users have grown accustomed.

4.2 Communication

The communication protocol of ports and messages is used at the fundamental level to organize Mach. Everything is an object addressed through messages by its port. Many tasks can send to a port, but only one can receive. Additionally, communication to the kernel and movement among processes is handled my messages. This provides excellent protection for users in an efficient manner, all by optimizing one area of the kernel. Efficiency is derived from avoiding copying messages from one task to another; Mach uses virtual memory remapping to send messages. This system is also useful for translating data between formats so that different hardwares and operating systems can communicate on top of Mach.

This solution has worked very well to solve the object oriented design, multiprocessing, and multiple-system goals of Mach. As this was the core design principle of the system, there are less issues with this solution. A possible issue with this protocol is shoehorning future systems which do not operate well within these confines.

4.3 Threads

Mach implements threads for each task to support multiprocessing and parallel computation. Mach's thread usage contributes to a range of scheduling difficulties arising from complexity. Research into managing multiple processors is generally new and difficult. Another issue is choosing a switching time quantum. However, Mach solves this issue using a time quantum which varies inversely with the total number of threads. Mach currently simplifies these complexities by keeping the scheduler at a thread level rather than a task level. Although issues arise with thread use, Mach has provided solutions at the cost of efficiency and complexity.

4.4 Memory

Mach is object-oriented, and uses memory objects to abstract data such as files or pipes. These memory objects are not typed or given importance, keeping then independent of the kernel. This allows the system to support multiple different memory models on top of the kernel.

Mach uses user level first and then kernel level memory managers for these objects. Programs wanting to access shared memory can use the same user level memory manager. Although this provides a great level of flexibility, efficiency is affected. A more intelligent kernel level manager could allow processes to operate on memory concurrently, given that distinct page areas are accessed. Because of the complexity, the user level managers often do not implement such logic. Furthermore, the virtual address space often has many regions with no data. Mach does not compress the space, even though crashes and failures could result. However, Mach currently is not limited by this, as address spaces are at least 4 GB.