What is MachUS

J. Mark Stevenson

28 July 1994

1. Introduction

Mach-US is a multi-server OS developed as part of the CMU MACH project. It has been developed by Dan Julin, myself and several others from CMU-Mach and OSF-RI. Its design goal is to provide a set of generic servers that can be used to support various standard OS's and to allow easier experimentation with OS interfaces.

The current release supplies a Mach2.5/4.3BSD API and runs most of the non-administrative UNIX application binaries that are provided as part of the i386Mach release.

2. Quick System Architecture Overview

Mach-US is a symmetric multi-server system. It has a set of separate servers supplying generalized system services (file systems, network server, process-mgr, tty server,...) and an emulation library loaded into each user process. This library uses the services to generate the semantics of an industry standard OS being emulated. Some of the system servers are, in part or whole, specific to the OS being emulated while most are meant to be fully generic. Thought was given when writing the OS specific ones to make parts of them reusable for other OS's. There is no "central" server, either for emulating a specific OS or for general traffic control. All multi-service actions are controlled by the emulation library.

A detailed explanation of the system and its design can be found in the paper entitled: "Generalized Emulation Services for Mach3.0: Overview, Experiences, Current Status" from the Nov.'91 Usenix Mach Symposium. This paper is available for anonymous FTP at "mach.cs.cmu.edu" in "doc/published/mach_us-multiserver.ps"

3. Mach-US Flexibility

The biggest single advantage of Mach-US is its flexibility. It offers a whole new, highly modifiable OS architecture without significant structural impediments to speed. This flexibility is accomplished by the following design features.

3.1. Object-Oriented Generic OS Interfaces

Several sets of C++ based object-oriented interfaces (virtual classes/methods for multiple inheritance) define the semantics supplied by the system servers. These interfaces are: access mediation, naming, I/O, net_control(OSI-XTI based), and async notification. The various servers then support a combination of these interfaces to do their work and an emulation library uses these interfaces to emulate its target OS.

This uniformity of access makes it easy for new servers to supply additional functionality by sliding into the name-space under the known interfaces. Then that functionality is generally available to the system users through their pre-existing software (for example, "ls /servers/net/tcp" will diplay the active TCP sockets/connections).

Since the interfaces are object-oriented it is simple to make refinements of them via inheritance.

OS specific services (like the process-mgr), may supply specific interfaces within the same basic C++ virtual class model.

3.2. Modular Services

Different functionality is separated into various servers: configuration, authentication, pathname, diagnostic, pipenet, ufs, process-mgr, tty, and network (with a University of Arizona xKernel protocol engine). This separation makes makes it simplier to develop and debug OS services. One can also add and subtract services as needed for a given invocation of the system. Furthermore, a bug in one service doesn't crash or corrupt the entire system.

3.3. Object Library/Code Reuse

There is an extensive object library both for the support of the generic interfaces as well as specific classes to implement general server building blocks. Some of the functions that are provided are: client-server binding, name-space manipulation, mapped-files/shared-memory/page-objects, IO/bulk-data and protection. These classes enable faster prototyping of a new server and ease creation of servers from foreign code.

Two common cases of considerable complexity are handled by the following packages of interface and implemenation classes and macros:

- Remote Method Invocation: The remote method invocation system enables clients of the system services to simply invoke a method against an OS Item. The appropriate method is correctly invoked on the appropriate server. This occurs with the needed authentication and protection guarantees.
- Interrupts/Signals: This package enables server code to handle interrupts and UNIX signals in a consistent and relatively painless fashion. The remote invocation system handles informing a server when a invocation should be interrupted. The macros and routines provided by the interrupt package can be placed in the server and emulation code at points where they can enable interrupts to be handled punctually, correctly and atomically. This package has been used to port previously interrupt ignorant code into a server that responds correctly to UNIX signals. The portion of the interrupt system that the servers use is not specific to UNIX signals but works for any source of interrupts (pending invocation cancelations). Most of the emulation_lib side of this mechanism is either OS independent, or easily modifiable for a different signaling semantic.

4. What is the Mach-US status

4.1. Current functionality: What's running

- Self Hosting: Mach-US has been used to build itself.
- Test Software Development: It is used as the default platform for development of test software used to test Mach-US functionality.
- Andrew Benchmark: Runs.

- Parallel Compile Test: There is a version of the standard Mach compile-test (which just compiles a number of small programs) that runs many compile tests at the same time. Mach-US runs this test with heavy loads on uni-processor and shared memory multi-processor boxes.
- Day to day tools: Most of the common UNIX utilities we know and love (csh, vi, gnu-emacs, find, ...) are working with out problems.
- FTP(d)/Telnet(d)/Inetd: Are used regularly.

4.2. Current Reliability

Positive:

- Runs and works: One can log into this Unix system and do useful work today. Typing at a cshell, one has full tty and job control, pipes and signals, file access and access control. It is very much a real UNIX.
- Stays up indefinitely: There is no noticeable system rot. It does not crash in the night, or during average use.
- Handles heavy computation stress: The system has been stressed by various tests, self-hosting, and general use. It handles the stress well.

Negative:

- First time applications uncover bugs: When a large new bit of software runs, it may uncover either a bug or a small missing feature/ esoteric semantic.
- Untried code paths: There is some code or odd interface features, that has never been used and hence may not work.
- Different error values: Mach-US may return a different error or errno than Mach2.5 would have for the same error.

4.3. Speed

In general the CMU Mach single-server(Mach-UX) runs %10-%20 faster than Mach-US for common usage tests. Yet in a highly parallel task (parallel compile test) on our multi-processor Sequent box, Mach-US performs slightly better than Mach-UX. It has also been measured to run much faster than Mach-UX for FTP.

Because of the sophistication of our emulation lib, most syscalls (as defined by occurrence during test) do not make any calls to the servers, but instead are handled in the client process. By this property, it should be possible to make Mach-US run faster than the Mach-UX system in general. Furthermore, with the exception of remote method invocation, we have yet to find the time to meter it heavily to discover what slowdowns exist. Simple changes alone, driven by such metering, should make the system as fast or faster than Mach-UX.

Yet, there are other obvious places for speedup: bundling common server call sequences, shared libraries for server text, optimizing forking and signaling, and optimizing out some debugging mechanisms.

5. Release and Distribution

Mach-US is now distributed via SUP to any hosts that are enabled to SUP other mach3 collections. To get a fully functional system a site must have a Mach 3.0 license and the prerequiste 4.3BSD license.

There is also a license-free release available. It is missing the sections of the ufs and tty servers that are derived from BSD code as well as parts of the libmach3 library that are derived from libc.a. While this release of the code will not support UNIX applications, it is available for study or reuse. The object-interface library, the remote invocation method and the interrupt handling package including the task master server may be of real value to designers of other object-oriented distributed systems.

For details on SUP'ing and running the Mach-US system see the note "Installing and Running Mach-US" in the file us-install.{ps,doc} in this directory or FTP'able from public/doc/mach_us/us-install.{ps,doc}.

Thank you for your interest,

J. Mark Stevenson
Senior Research Programmer
jms@cs.cmu.edu

Table of Contents