Installing and Running Mach-US

Mary R. Thompson

27 October 1994

1. Introduction

The Mach-US multi-server system is comprised of a set separate servers supplying generalized system services(file systems, network server, process management, tty server ...) and an emulation library loaded into each user process. The currently distributed emulation library (bsd_all.lib) uses the services to generate the semantics of BSD 4.3 Unix. The servers themselves are written to use the services provided by the Mach micro-kernel and the other servers.

This document will describe two ways to start up the multi servers. It will also explain the file system layout and scripts that we use.

2. Overview

In addition to the servers that provide OS services, there are three support servers needed to get things started: a bootstrap server, the configuration server and the diagnostic server. The bootstrap server is the user-level program that the Mach micro-kernel started. It must provide enough Unix functionality to support a shell script *exec'ing* Unix load format files from a Unix file system. The configuration server starts the rest of the servers and sets up some connections between them. The diagnostic server is used by the servers to write out error messages. Depending on its configuration it will use Unix style write calls or call the micro-kernel device_write interface on the console device.

Either the UX server or the POE server can be used as the bootstrap server. The only function this server needs to provide is the exec'ing of the various programs that comprise the multi-server, output onto stderr and input from files or stdin. Once the servers have been started and if they were built in a stand-alone configuration, they should not make any further use of the bootstrap server.

Starting the multi-servers from the UX server has the advantage of having a complete stable Unix system running on the machine which can be used to debug the multi-server servers. It has the disadvantage of having to partition the disks so that only the Unix server or the multi-server UFS server can write on any given file-system. Normally this is accomplished by setting aside one or more partitions that are not mounted by the UX server and are mounted read-write by the multi-server, and then mounting UX system partitions read-only by the multi-server. (Note: there is no coordination between US and UX file systems, when a partition is mounted in both systems, writing it in one system and reading it from the other can lead to unexpected results, writing from both can cause filesystem corruption.)

The setup procedure described here assumes that you have been running Mach 3.0 with a Unix single server and want to continue to run a single server system either in alternation with a stand-alone multi-server system or alongside the multi-server. We have chosen to have a different root partition for each system, in order to allow the systems to each write on their own root partition. Most of the directories and files on the root partitions can be shared via symbolic links but a few must be different.

3. Hardware and System Prerequisites

Mach-US runs on a Mach 3.0 kernel and supports the I386Mach-BSD4.3-Tahoe API that CMU distributes to licensed sites. All of our developement and testing has been done on i386/483 ATbus machines. While there is not much machine specific code in Mach-US, there is some assembly language code that would need to be added before it could be run on a different architecture.

Most of our recent testing has been done on the MK83 release of the micro-kernel. Mach-US has a rather different pattern of kernel use than the UX server and has often found kernel bugs that UX does not. Therefore, we recommend caution in using it with other kernel releases.

Running the multi-server on a machine with less than 16Megabytes of memory is not recommended. In our most recent experience of running the stripped servers, standalone on a machine with 12M of memory, too much time was spent paging.

I386 machines do not always gracefully handle more than 16M of memory. The more recent Mach kernels can be patched to use all the available memory if they do not have SCSI disk controllers. The kernel variable use_all_mem needs to be set to 1.

4. Initial Setup Procedure

4.1. SUP'ing Mach_US sources and executables

If your host has been enabled to SUP any of the licensed mach3 collections from CMU, the sources and executables for Mach_US also can be SUP'ed using the following SUP command line.

```
mach3.us release=default host=x29.mach.cs.cmu.edu hostbase=/usr2
      base=<your-base-directory>/us crypt=<your_crypt>
```

See the note in ../public/FAQ/mach3_supinfo if you need more information about SUP'ing Mach3 collections.

4.2. Install the multi-server executables

You can get a copy of the multi-server executables either by building them from the sources or taking them from the release directory (.../src/us) SUP'ed as part of the mach3.us collection.

If you are building from the sources, after the `build_all` pass is completed, you need to run `odemake` again with `install_all` target. This pass will copy the libraries, servers, programs and scripts that were built into a release directory specified by the make variable `TOSTAGE`. In order to match the default names coded into the configuration server this release directory should be either named or linked to by the name `/mach_servers/us`. There is no easy way to change this name. Even though you may have specified an alternative directory name for the startup program during the boot sequence, that name is not passed through to the multi-server. This directory must be readable by the bootstrap server. It should not be on the multi-server root partition.

There are two different configurations for the tty_server. One is built with the MACH_US flag, is named tty_server.us and is used when the system is started with POE. The other is built with the MACH_VUS flag, is named tty_server.vus and is used when the system is started by UX. Whichever one you want to use must be linked to by the name `/mach_servers/us/bin/tty_server`.

The emulation library `lib/bsd_all.lib` must be copied to the directory `/lib` on the multi-server root partition. The script described in the next section does this for you, but if you subsequently install a new version of this file you must remember to copy it to multi-server `/lib` directory.

4.3. Create a root partition for the multi-server

You need a partition with an empty file system that is not flagged for mounting by /etc/fstab. There is a C-shell script **us-setup** that will setup this empty partition to be the multi-server root partition. This script mostly creates symbolic links from the multi-server root partition to the directories on the single-server root. It creates new directories for `/dev` since devices are handled rather differently by the multi-server, and for `/tmp` and `/usr` so that the multi-server can write in those directories when it is running alongside the single-server. The `/etc` and `/lib` directories are new directories that are populated with symlinks back to the single server disk partitions for most files. The file `bsd_all.lib` is copied in to the `/lib` directory and the file `inetd.conf` in the `/etc` directory is considerably simplified from a standard Unix one.

Some of the links to file system partitions are special "TRANSPARENT" links. These links point to "prefix names". References to these names will be mapped through the prefix_table to the UFS server that is handling access for that partition. So the names used in the us-setup script for "TRANSPARANT" links must agree with the names set in the `prefix.config` table. See the section "A note about Name Spaces" for more explanation of the prefix table.

A template for this script is released as `etc/us_setup.template`. It should be renamed to us_setup and edited to reflect the actual names of the disk partitions on your machine. When running this script, you should be *cd'ed* to the empty partition that you wish to initialize.

At this point the "/usr" links created to user home directories of Mach-US users should be replaced with actual directories in the /usr directory on the multi-server root partition.

4.4. Edit the host specific configuration files

There are four files in addition to the us_script that are used to provide information about the local host. Templates for these files are released into the directory **etc**. They should be copied from <file>.template to <file> and then edited to reflect your local environment. The template files assume the host has three partitions `/, /usr, /usr1` that are normally mounted when running the Unix server and that the multi-server is to have read-only access to these partitions.

prefix.config Used by the configuration server to initialize the emulation prefix table. Contains the mapping of name prefixes to the server that handles the objects starting with that prefix. It needs to be edited to reflect the UFS partitions that you have on your host.

rc.us A shell script run by the configuration server to start all the other servers. This script needs to be edited to start a UFS server for each partition that you want the multi-server to access

STARTUP.fsadmin
Mounts the servers in the pathname server's name space. Needs to be edited to mount the UFS servers that have been started by `rc.us`.

start_net Run to start the network server. Needs to have the local host's IP address, Ethernet

net device and gateway address.

init_env.template Can be "sourced" by the first multi-server process to initialize its environment. Edit it to set whatever PATH's and aliases you feel comfortable with. It can be copied to the multi-server root partition since that is where the first process starts up.

5. The Bootstrap Process

When the Mach micro-kernel is booted it will start one user level server. By default this is the program it finds in `/mach_servers/startup`. This document describes the cases where the startup server is either the UX server or the POE server. See **Setup for Mach 3.0**, (.../public/doc/unpublished/mach3_setup.{ps,doc}), for details of how to use a directory other than `/mach_servers` so that you can choose the startup program during the boot sequence. Both of POE and UX will look for a script /<mach_servers_dir>/`rc` to execute before starting up a user shell.

5.1. Starting the multi-server from POE

1. Choose `/poe_servers` as the name of your alternative ***mach_servers_dir***. Make sure the `startup` program is POE rather than the UX server and that poe_emulator and poe_init are in place. See the document **A Brief Description of the POE server**, (.../public/doc/unpublished/poe_notes.{ps,doc}, for all the details on how to boot POE and how to have alternative mach_servers directories.

2. Place the following script in `/poe_servers/rc`

```
echo Starting POE rc
/etc/fsck -p
echo About to mount
/etc/mount -a
echo starting US
/mach_servers/us/bin/config_server \
        /mach_servers/us/bin/emul_init \
        -t /dev/console /bin/csh
echo POE.rc complete
```

   Note that the `/etc/mount -a` should not cause the multi-server root partition to be mounted.

3. Be sure that you have the MACH_US configuration of the tty_server installed as `/mach_servers/us/bin/tty_server`.

4. Reboot the system.

5. When it gives you a prompt, you will be talking to a */bin/csh* that is a multi-server client and thus any program you start will be a multi-server client.

6. At this point you are running as the super-root and can source the *init_env* script to set up the environment or you can run */bin/login* and become any user that has an account on the machine.

5.2. Starting the multi-server from UX

1. Be sure that you have the MACH_VUS configuration of the tty_server installed as `/mach_servers/us/bin/tty_server`.

2. Create a window to run the multi-server process in. In that window type

```
tty
```

3. remember the tty device that was returned and type

```
sleep 90000
```

This reserves this window/tty for use by the multi-server processes.

4. In another window type

   ```
   /mach_servers/us/bin/multi <tty> &
   ```

   Where <tty> is the value was returned by the "tty" command. This tells emul_init to use the other window as the tty for the multi-server process. **multi** is a script that arranges to log its output, kills off any previous multi-server incarnation (via the **kemul** script), and starts the configuration_server with an emulation command line to run emul_init. The user that runs the multi script must be able to *fsck* the multi-server's partition. e.g `/dev/rhd0f` and must also be able to get the device port from the mach kernel. e.g. either uid=root(0) or gid=kmem(2). The output from this command will be logged in `~/multi.log` unless the -l <logfile> option is used to chose a different name.

5. As the result of the multi command a prompt will appear in the first window. This is from a multi-server *csh*. Any processes started up in this new *csh* will be running a multi-server client as root.

6. At this point you are running as the super-root and can source the *init_env* script to set up the environment or you can run */bin/login* and become any user that has an account on the machine.

5.3. Bootstrapping details

The POE `rc` script or the UX servers `multi` command runs the configuration server, `bin/config_server` to start the rest of the multi servers. The command line syntax for the configuration is of the form:

```
config_server [-d<debug level>] [-p<prefix file>]
              [-s<startup script>] [emulation command line]
```

With no arguments the configuration server will start the servers by running the C-shell script, `/mach_servers/us/etc/rc.us`, fork a C thread to do port name service, initialize a prefix table for names used by the emulation library from `/mach_servers/us/etc/prefix.config`, and execute `/mach_servers/us/bin/emul_init` which starts the first multi-user process on the console. This default behavior assumes you are bringing up the multi-server environment stand-alone, i.e. started by POE.

You can specify a prefix file other than /mach_servers/us/etc/prefix.config by using the "-p" option. Using the "-s" option, you can specify a server startup script other than /mach_servers/us/etc/rc.us.

If you just want to start the servers and not the emulation, you can specify the emulation command line to be something innocuous like "/bin/csh". You can then run the emulation command separately in a shell where the configuration server is providing port name service.

In order to satisfy the default names, `/mach_servers/us` should be a link to the directory containing the multi-server executables. The directory must be on a partition readable by the bootstrap server.

Once all the servers have been started, the `rc.us` script runs the program `/mach_servers/us/bin/fsadmin` with `/mach_servers/us/etc/STARTUP.fsadmin` as the input script. This program mounts the various servers with the pathname server. **fsadmin** is a command interpreter which makes calls directly on the multi-servers. It understands about 2 dozen commands in addition to `mount` and can be used for testing and debugging the servers.

After the `rc.us` script has been completed, the configuration server will fork. The parent process remains to provide port name service via the snames/netname interface and the child process *exec's* the emulation command. If this command is `/mach_servers/us/bin/emul_init`, the process will replace the bootstrap server's emulation library with the multi-server emulation library, thus transforming the task into a multi-server task. It then *exec's* whatever is left on the emul command line that was passed to the configuration server. Normally this is */bin/sh, /bin/csh* or */bin/login*. This instantiation of the shell will be a multi-server client and thus any programs that it runs will be also be multi-server clients, i.e. loaded with the multi-server emulation library.

6. A note about Name Spaces

There are several disjoint name spaces maintained in the multi-server environment. The first one is the **port name space** maintained by the configuration server. This consists of a mapping of ports to server names. These names are known by convention to other servers or programs. The interface to this service is defined by the calls *netname_check_in* and *netname_look_up* as specified in *netname.h*. These calls are made on the predefined port *name_server_port*.

The second name space is the **pathname space**. This space is maintained by the pathname server with the ns_resolve interface that maps a pathname to a proxy for the object.

There is a third name space defined by a **prefix table** maintained by the emulation library. This table is initialized by the configuration server with some "system" entries defined in the `prefix.config` file. The prefix table maps the beginning of path names to the server that handles names starting with that prefix. Unix-style references to pathnames are handled by the emulation library which strips off the longest prefix it knows about and hands the rest of the name to the appropriate server for an ns_resolve.

Most servers maintain a name space for the objects they manage. There is a UFS server for every mounted partition, the tty server has a name entry for each potential tty or pty, the task_master has a name space for all active tasks, and the pipenet server has a name space for any active pipes. These name spaces can be accessed through the **/servers** directory.

In the current configuration setup, all the servers that the configuration server starts are entered in the port name space, (see `rc.us` for the servers and their names). The pathname server and the UFS servers are given initial entries in the prefix table (see `prefix.config`). The UFS servers, tty_server, task_master, pipenet_server and net_server are all given entries in the pathname space in the directory `/servers`. (see `STARTUP.fsadmin`). The UFS servers and pathname server are given *transparent symlinks* in the ***/*** directory in order to make this directory look more like a traditional Unix root directory.

7. A Note about Building the Mach_US tree

The programs in the Mach_US tree are built with the same tools and procedures as the rest the Mach 3.0 system. The tools are available in the mach3.release collection. A description of the tools and build procedure is given in the document **Building Mach 3.0** (.../public/doc/unpublished/mach3_build.{ps,doc}).

There are a few variations specific to this tree. There is an additional make rules file called osf.cmu_machus.std which defines the compile variables for c++ files as well as a few compilation constants that are unique to the multi-server. This file is distributed in the directory `src/mach_us/mk` and is exported to the standard place: `export/@sys/lib/mk`

In order to provide stability and independence from the other source trees, the mach_us tree builds with its own copy of the kernel and unix include files. These include files and the libraries are exported to the directories `export/@sys/us/{include,unix_include,lib}`. These directories are searched during the build before the standard export directories, `export/@sys/{include,lib}`. The libraries are distributed in the `release/@sys/us/lib` directory. If you are doing a shadow build you can either add this directory to your LPATH after you run the setvar script or move these libraries up to `release/@sys/lib`.

When doing a shadow build, you have to go though the include part of the export pass to populate your local `export/@sys/us/{include,unix_include}` directories before attempting to build any programs. To do this run `odemake export_all` and kill it when it starts building libraries.

In looking at the Makefiles or code, you may notice a CONFIGURATION build variable which is used to select the version of LIBMACH3 to use and is passed as a -D flag to cc. Currently there are two different configurations for servers:

MACH3_US — for components that must operate directly on top of the Mach 3.0 micro-kernel with no access to Unix system calls but with access to all normal services of the multi-servers.

MACH3_VUS — for components that normally operate on top of the Mach 3.0 kernel, but have access to Unix system calls as well as the multi-servers.

The effective difference between MACH3_US and MACH3_VUS is in the handling of I/0. MACH3_US programs use the diag_server to handle output (normally for logging error messages) and do not do any input. MACH3_VUS load with the stdio routines from libc and thus end up doing read and write syscalls. These traps are handled by the bootstrap server that was used to load the multi-server.

The version of the stdio routines that are used is a function of the LIBMACH3 library that is loaded. LIBMACH3 defined as "libmach3_sa.a" for the MACH3_US configuration. This library includes a version of *_filbuf* that returns EOF and *_flsbuf* that calls the diag server. It also provides versions of the non-syscall parts of libc.a.

LIBMACH3 is defined as "libmach3_vus.a -lc" for the MACH3_VUS configurations. libmach3_vus.a has no version of *_filbuf* and a version of *_flsbuf* that makes a write syscall. Since it is loading with the normal libc.a, the syscalls are handled via the emulation library by the bootstrap server, i.e. the server that loaded the program.

Applications written to run on a multi-server system should either link with libmach3_vus -lc if they want to use Unix, multi-server and Mach 3 features, or just -lc if they only want to use generic Unix features. These programs when run by a multi-server shell will be loaded with the multi-server emulation library. This library directs Unix syscalls to the appropriate multi-server function.

8. Useful Programs

There are some useful programs (other than the system servers) in the .../release/us/bin directory:

- **Fsadmin** can be run within Mach-US as well as at setup. It can be used for directly querying the various servers directly (without the interposition of the emulation lib). It is an interactive program and help is included.
- **Us_ps** is run within Mach-US to list the process ids and their exec strings. More process info can be found by running "ls" in the `/servers/tm/*` directories.
- **Gdb**: A version that works better for attaching to processes and continuing them. This is run under the UX server, one may attach to a process by machid (from the ms program) by using the negative of the id value.
- **Fsck**: The default version supplied with the CMU TahoeBSD release prints out messages warning about fast symlinks. This version does not.

Table of Contents