Building Mach 3.0

Mary R.Thompson

13 July 1993

1. Introduction

We have now switched our Makefiles to use the OSF/ODE make program and OSF style Makefiles. This make program is basically the one by Adam De Boor that was released on the BSD 4.3-Reno tape. The OSF modified it to support CMU style object and shadow directories. The version we are using came from the OSF DCE v 2.1 tape. The programs are all available without license restrictions and have been placed in the *buildtools/ode* directory. Currently they are used to build themselves, the mk, ux, user and BNR-server (bsdss). We have also switched to use gcc 2.3.3 for all our compilations. Sources for this compiler are in *buildtools/gnu*. You can build it yourself and install it where you want it or you can use the binary version we provide in the mach3.release collection. External users need to setenv GCC_EXEC_PREFIX <your_installation_dir>/@sys/lib/lib-gcc/, since otherwise gcc will try to find it's components in */afs/cs.cmu.edu/project/mach3/latest/release/@sys/lib/lib-gcc/*, followed by */usr/local/lib/gcc-lib* and */usr/lib/gcc* .

Once all the tools are installed, the procedure for building components of the system is to run a script that sets a number of environment variables that the Makefiles depend on, then switch to the directory containing the sources of the target to be built, and type *odemake*.

There are several documents to help you understand how this process works. There are OSF man pages on make.1 and makefiles.5, a man page on the additional makefiles for the mach3 project, a man page for setvar.1 and a postscript tutorial by Adam de Boor on the make program.

2. Bootstraping the ODE tools

This section can be skipped if binary versions of the following tools are already available: *gcc 2.3.3, odemake, genpath, makepath, md, release, wh* and *setvar*. At CMU these files are in /afs/cs/project/mach3/{latest,alpha}/release/@sys. External to CMU they can be obtained by SUP'ing the mach3.release collection.

The ODE tools were designed by the OSF to be portable to different machines and versions of Unix. The first step in this process is to build the tools themselves. A script (or set of scripts) has been provided for this purpose. The first script is *setup.sh* which we have placed in the *buildtools/ode/setup* directory. It is called with a *context* argument which defines the machine and host OS, in our case it is one of *i386_mach, i386_bnr, pmax_mach, pmax_ultrix, sun3_mach, sun4_mach, vax_mach, luna88k_mach* or *pmax_mach_X_alpha*. The *setup.sh* script runs a sub-script, *<context>/setup.sh* which is used to set the machine and OS specific variables. If you are trying to build the tools on a platform other than one of these you should create a customized script for your new context.

The location and procedure for running the setup script is pretty inflexible since it needs to set up a directory structure and know where the sources are. The directory structure (which may be rooted wherever you wish) that it assumes is as follows:

```
src/{buildtools,mk,ux,user,bsdss}/...
```

```
src/buildtools/ode/{setup,mk,bin,lib}
obj/<context>/{buildtools,mk,ux,user,bsdss}/...
release/<context>/{bin,lib,include,etc,man,special}/...
export/<context>/{bin,include,lib,special}/...
```

Actually the *setup.sh* script only cares about the source directory names in the ode subdirectory and the location of the object directory. There must be a *Makeconf* file in the *src/buildtools* directory that specifies *MAKEOBJDIR* to match where *setup.sh* believes it is.

To run the setup script you should

```
cd src/buildtools/ode/setup
sh -x setup.sh <context> >& log &
```

The script will create all the *obj, release* and *export* directories that it needs, use the *bin/make/bootstrap.sh* script to build make, install make in *export/<context>/bin/odemake*, which it has added to *PATH*. The *bootstrap.sh* script defines some variables dependent on the *context*. If you are building in a new *context* you will need to modify this script. It then uses this *odemake* and the Makefiles in *ode/setup/mk* to build *genpath, makepath, release, md* and *wh* and to install them in *export/<context>/bin*.

The version of gcc 2.3.3 we provide is the standard gnu release, with some configuration changes for Mach. The gnu directory contains scripts and instructions on how to do build it.

At this point you have the bootstrap tools that are needed to build the rest of the system. You still need to have the standard Unix tools: *csh, sed, yacc, lex, awk, tar* and an assembler and loader available. Now you should run the setvar script and build the complete buildtools directory. This will rebuild the odetools plus a few additional things depending on what platform you are building on.

3. Setting the environment for odemake

3.1. setvar

The ODE make and makefiles depend on certain environment variables. A set of scripts, *buildtools/ode/setup/{setvar.csh,<context>/setvar.csh}* have been provided which make sure that all the necessary target directories exist, exec a shell and set the necessary variables.

There are three different situations in which the build may take place.

1. Everything is in the local source, export or release area. This includes all the tools we provide and whatever exported includes or libs are needed.
2. The local area contains all the sources of the programs that are to be built, but tools and includes or libraries from other areas are to be found in a complete system release area.
3. The local area is a shadow of some system release area, and contains only the sources that are different from the originals. In this case sources as well as tools, libs and includes are to come from the system release area, unless they exist in the local area.

setvar can be called with a variety of arguments to deal with these cases.

```
setvar.csh [-basedir <dir>] [-masterbase <dir> ]
           [-systembase <dir> ] [-makesyspath <dir>]
           [-target <machine>] <host_context>
```

The options are:

-basedir <dir>: the base of the tree which contains the local src, obj and export directories. Defaults to the parent of a superior *src* directory.

-masterbase <dir>: when shadowing set this to the base of the source and release directory that is being shadowed.

-systembase <dir>: is the base of the tree that contains a current system release. When shadowing a fully built release area this variable does not need to be set, since anything that is not in the local area should be found in either the master/release area or on the default PATH. If not shadowing and this variable is not set, then a bootstrap script should be run to get the the system tools and include files generated and released into the local export directory.

-makesyspath <dir>:
is a list of directories (separated by ':'s) that is searched by odemake to find the .mk rules files. Defaults to <basedir>/export/<host_context>/lib/mk, followed by masterbase/release if shadowing, and systembase if it was given.

-target <machine>: should only be used when cross-building and the target_machine is different from the host_machine. Otherwise target_machine is set from the host_context

-savepath: if set, the current PATH will be added to the end of the PATH that is set here. By default PATH will only contain the minimum necessary to build.

-help: prints a brief help message

host_context: is host_machine_os specification that selects sub setvar.csh that sets machine specific variables such as HOST_MACHINE, target_machine, target_cpu and default search paths.

For example:

Case 1: where everything is local in the directory /mach3/{src,export,release}

```
cd /mach3/src
/mach3/export/i386_mach/etc/setup/setvar i386_mach
```

Case 2: /usr/mrt/mach3 is contains all the sources for a kernel build but other tools are to be found in the system release area, /afs/cs/project/mach3/latest/release.

```
cd /usr/mrt/mach3/src
alias sv ‘/afs/cs/project/mach3/latest/release/@sys/etc/setup/setvar‘
sv -systembase /afs/cs/project/mach3/latest/release/@sys i386_mach
```

Case 3: /mach3 is a local shadow area for a kernel build and the master area is /afs/cs/project/mach3/latest.

```
alias sv ‘/afs/cs/project/mach3/latest/release/@sys/etc/setup/setvar‘
sv -basedir /mach3 -masterbase /afs/cs/project/mach3/latest i386_mach
```

3.2. Environment variables

The following environment variables are set by *context*/setvar.csh

HOST_MACHINE,TARGET_MACHINE,target_machine,TARGET_CPU,target_cpu
Type of machine.

host_context Host machine and operating system.

TARGET_OS,target_os
Set to mach.

DEFCPATH Default path to search for include files.

OBJECT_FORMAT Format of object files. e.g A_OUT, COFF, MACH_O

ARCHIVE_FORMAT
Format of archive files. e.g. BSDARCH,

Some flags for porting the ODE tools

The following environment variables are set by setvar.csh

PATH Path to be searched for binaries. This is the export directory, the release directory and whatever DEFPATH was set for this system.

CPATH Path to be searched for include files. Used by the C preprocessor.

LPATH Path to be searched to find crt0.o and migcom.

INCDIRS A list of -Idirs to be passed to the pre-processor instead of relying on *CPATH*.

LIBDIRS A list of -Ldirs to be passed to the loader instead of relying on *LPATH*.

PROJECT_NAME,project_name
Selects the project specific make rules by causing the inclusion of osf.<project_name>.mk into the makefiles. Set to mach3 for all the Mach 3.0 collections.

RULES_MK The name of the stardard make rules file to start the includes in. It is set to osf.std.mk which is looked for in the MAKESYSPATH directory. This is used instead of the BSD standard sys.mk.

EXPORTBASE, SOURCEBASE, OBJECTDIR
Bases of the respective trees. SOURCEBASE and OBJECTDIR are used by *genpath*. OBJECTDIR is also used by Makeconf.

SOURCEDIR Can be set to define a backing directory for the sources. Used by genpath and Makeconf

In an OSF environment, the *setvar.csh* script is replaced by rc-files and the *workon* program.

4. Make variables

4.1. Programs and Flags

The *odemake* envrionment uses a set of standard rule files that defines many variables as well as standard rules. These files are included by the `.include <${RULES_MK}>` in each Makefile. *${RULES_MK}* is defined as *osf.rules.mk* and is found on the *$MAKESYSPATH* path which is *{export,release}/<context>/lib/mk*. The majority of variables are defined in *osf.<projectname>.mk*, in our case *osf.mach3.mk* and *osf.std.mk*. The rest of the *.mk* files are rules. In general, these *.mk* files define variables only if they have not been previously defined which allows you to overide these values with environment variables or by setting them in Makefiles. The values are often fuctions of other variables set in the environment or Makefiles.

Variables are defined for most programs that might be used by rules: e.g. _CC_, _LD_, RM, RANLIB etc, and for default flags used by the rules. Many of the flags follow the pattern of combining the following catagories of flags:

DEF_xFLAGS Set conditionally in *osf.mach3.mk*.

${.TARGET}_xFLAGS or xFLAGS
Set in the Makefile.

${.TARGET}_xENV or xENV
Set in the environment.

${.TARGET}_xARGS or xARGS
Set on the command line.

4.2. Variables defined in Makeconf

The *Makeconf* file is read by *odemake* before any other Makefiles and sets a few basic variables. Its location is used by *odemake* to determine two other variables.

CONFIG Selects what configuration options are chosen for the kernel and Unix server. Defined in *Makeconf*.

MAKEOBJDIR The directory in which the objects are placed. If this is undefined the objects are put in the source directory. Defined in *Makeconf* as a function of *OBJECTDIR*

MAKESRCPATH The base of the Master sources if you are shadowing. Set in *Makeconf* if *SOURCEDIR* is defined.

MAKECONF Defined by *odemake* as the absolute pathname of the *Makeconf* file that it is using. This value is used in our Makefiles to determine the base of the source and/or object,release trees. Exported to the environment.

MAKETOP Defined by *odemake* as the relative pathname from the directory in which *odemake* was invoked to the directory containing the *Makeconf* file.

4.3. Search path variables

odemake uses an agumented version of VPATH to search for source files. It gives -I flags to *cc* and *mig* to search for header files, -I flags to *md* to shorten dependencies and -L flags to *ld* to search for libraries. The variables that are used to set these values are:

VPATH The search path that *odemake* uses to find the prerequisite files when it is evaluating a dependency rule. This may be set by individual Makefiles, but *odemake* expands any given names to include the same names in the source tree and any backing source areas. (See Shadowing) If a file name starts with "./" only the current directory, which is normally in the object tree, is searched.

INCDIRS and LIBDIRS
set by *setvar* to the local export directory and and master or system release area. These directories are not expanded.

INCFLAGS and LIBFLAGS
set in Makefiles and expanded by *genpath* relative to the local source area, and any master source area.

4.4. Building on a DecStation Ultrix system

. The context *pmax_ultrix* has been defined to setup the environment to allow our sources to be built on a DecStation running Ultrix 4.2.

You may need to use gcc 2.3.3 to build our sources without errors. The Mips compiler version 2.1 is not sufficiently ANSI-complient to work. I have not tried any later versions. The binary verison of gcc that comes in the mach3.release release=mips will run on an Ultrix system.

The procedure that has been tested on a DecStation runing Ultrix 4.2 at CMU is as follows:

- SUP the mach3.buildtools, mach3.kernel, mach3.unix and mach3.release, release=mips collections. Most of the binaries in the mach3.release collection will only work after you are running a Mach kernel. But the buildtools: gcc, odemake, makepath, genpath, md, wh and xstrip will work under Ultrix as well.
- Modify *src/buildtools/ode/setup/pmax_ultrix/setvar.csh* to have GCC_EXEC_PREFIX, and DEFCPATH point to where you plan to install gcc-lib. I suggest *release/pmax_ultrix/lib/gcc-lib/*. (see below)
- Add the line

  ```
  setenv KERNEL_PMAX_CONFIG STD+OTHER
  ```

  to *src/buildtools/ode/setup/pmax_ultrix/setvar.csh* if you have SUP'ed the mipsfree release of mach3.kernel.
- Use the script *src/buildtools/ode/setup/setup.sh* with the context *pmax_ultrix*. This will build and export to *export/pmax_ultrix/bin* the buildtools. The native compiler, include files and libc will be used for this build. This step will also export the odemake rules files (*.mk) and the setvar scripts.
- cd to *release/pmax_ultrix/bin* and create a symlink there to the gcc that was SUP'ed from mach3.release. These files should be in *release/pmax_mach/bin*.
- cd to *release/pmax_ultrix/lib* and create a symlink there to *../../pmax_mach/lib/gcc-lib*,
- Now run the setvar script with context set to *pmax_ultrix*. You can run the script either from the export area or the source area, but if you make modifications to it, remember that there are two copies. The source version can be re-exported, by cd'ing to *src/buildtools/ode/setup* and typing "odemake".

At this point you should have your paths set to use the buildtools from *export/pmax_ultrix/bin* and the compiler from *release/pmax_ultrix/{bin,lib}*. As you start building, include files and libraries will be put in *export/pmax_ultrix*.

Now build the mk and ux sources as described elsewhere in this document.

4.5. Building on a BNR2 derived system

The context *i386_bnr* has been defined to facilitate building the Mach sources on a system running a BNR derived system, in our case Jolitz's BSD386 with a gcc 1.39 compiler. There is also a release=i386bnr of the mach3.release collection, that provides the buildtools binaries as well as a kernel and BSDSS server.

If you SUP the mach3.release collection, the buildtools, including cross versions of ld(x), ranlib(x) and ar(x) will be put in the *release/i386_bnr* area. As long as your release area is relative to the same base as your sources, when you run the setvar script, your paths will be set to use the files in the release area, unless they are superceded by ones you have built in the export area.

The BDSSS server, in place of Unix server, supports the BNR binaries.

If you are interested in building the sources in a "pure BNR" manner, e.g using the local version of make, rather than odemake, there is a script and some instructions in the BSDSS source collection.

5. Building from a set of sources

5.1. Overview

The basic procedure for building from a set of Mach 3.0 sources is to execute the *setvar.csh* script, cd to the directory in which the sources are to be found and type *odemake*.

odemake will descend down the subtree as specified in the *EXP{BIN,INC,LIB}_SUBDIRS* variables in the Makefile, first building and exporting any export targets, and then into the *SUBDIRS* list building the rest of the targets. Once all the exported targets are in place, individual targets can be built by *cd*'ing to the source directory and typing *odemake* <*target*>.

The target *export_all* will do only the exporting passes, and the target *export_<file>* will export a single file or program. The target *install_all* along with a *TOSTAGE* argument will release everything that has been previously built to the *TOSTAGE* area. See the man page makefiles(5) for more possible targets.

The following sections describe in detail the build steps for the micro-kernel (mk tree) and the unix-server (ux tree). You do not really need to understand this unless you want to modify these programs.

5.2. Building the mk tree

The make procedure for the *mk* tree is more complicated than a build of a tree of simple unrelated programs. The *mk/Makefile* has been set up to build and install the pieces of this tree in the order in which they are needed. If you are not absolutely sure of what you are doing you should always start the *odemake* in the directory *mk*. If you start off in one of the subdirectories you may either find that files are missing or you are getting incorrect versions of files from incorrect directories. The building of the kernel itself is also complicated by the fact that it needs to be dynamically configured for different hardware platforms and configurations. This is accomplished following the tradition of BSD 4.3 of having the *config* program write the Makefile, named *Makefile.internal*, that will be used to build the kernel.

Mach has the added complication that the bootable kernel is built in two pieces: the kernel itself named *mach_kernel* and the default pager named *bootstrap*. Another program, *makeboot*, binds the two files together into a single load image named *mach.boot* as the final step of the kernel build.

The layout of the mk source tree is as follows:

`bootstrap`	Makefile and sources to build the default pager.
`conf`	a link to `kernel/conf`.
`include`	contains the Makefiles to build, export and install the *mk* collection include files that are used outside of kernel builds.
`user/libmach`	Makefile and sources to build crt0.o, libmach.a and libmach_sa.a
`user/threads`	Makefile and sources to build libthreads.a
`kernel/src/{config,doconf,makeboot,mig}`	Makefiles and sources to build each of these programs which are needed to build the kernel

`kernel/conf` The Makefile templates and configuration files that *config* uses to create the Makefile for the kernel, *Makefile.internal*.

`kernel` The Makefile that starts off the building of the kernel and all the kernel sources. It ends up switching to the configuration directory and calling *odemake* with the newly created *Makefile.internal* as the Makefile that actually compiles files for the kernel.

When you do a *odemake* in the *mk* directory, it will proceed to make and export all of the export targets in each of the subdirectories in the *EXPDIRS* list in each Makefile in the order in which they appear. If you wish to start the make at any other place in this tree, you need to be sure that all the prerequisite tools, libraries and includes have been exported.

In case you are curious what all the steps are doing this a summary of the major ones:

- **cd kernel/src/mig/include; odemake MAKEFILE_PASS=FIRST build_all**

  exports the include files that are needed to build mig.

- **kernel/src/mig**

  builds and exports mig.

- **cd include; odemake MAKEFILE_PASS=FIRST build_all**

  exports all the kernel include files that are needed by external programs

- **cd user/libmach; odemake MAKEFILE_PASS=FIRST build_all**

  builds and exports crt0.o, libmach.a and libmach_sa.a

- **cd user/threads; odemake MAKEFILE_PASS=FIRST build_all**

  builds and exports libthreads.a

- **cd makeboot; odemake MAKEFILE_PASS=SECOND build_all**

  builds and exports the default pager,

- **cd mk/kernel/src/makeboot; make MAKEFILE_PASS=SECOND build_all**

  build and exports makeboot program

- **cd mk/kernel/src/config; make MAKEFILE_PASS=SECOND build_all**

  build and exports config program

- **cd mk/kernel/src/doconf; make MAKEFILE_PASS=SECOND build_all**

  exports doconf script

- **cd kernel/conf; odemake MAKEFILE_PASS=BASIC build_all**

  Runs the Makefile that is in the conf directory, which runs doconf to generate the <CONF> file from MASTER, MASTER.local, <target_cpu>/{MASTER,MASTER.local} Then cds<CONFIG>; runs the config program which reads the <CONF> file, conf/files, conf/<target_cpu>/files. conf/template.mk, conf/<target_cpu>/template.mk and generates the options .h files and the Makefile.internal in the object directory.

- **cd <CONFIG>; make MAKEFILE_PASS=BASIC build_all**

  runs the generated Makefile that builds the kernel.

There are several shortcut targets that can be used *carefully* during repetitive kernel builds.

- **cd mk/kernel; odemake kernel**

  exports any kernel headers you might have changed and rebuilds the kernel.

- **cd mk/kernel; odemake kernel_only**

  only builds the kernel, does not worry about changes to header files.

- **cd mk/kernel/<CONFIG>; odemake file.o**

  Can be used after you create an empty directory with the CONFIG name to recompile one file.

- **cd mk/kernel/<CONFIG>; odemake relink**

  Can be used after recompiling a few .o files. Does not check for any out-of-date dependencies.

5.3. Building the *uk* collection

Building in the *ux* tree has some of the same complications as the *mk* tree. Again you should start your make from the *ux* directory, to ensure that things are done in the proper order. Here it is only important that the *include/Makefile* installs the *.h* files before the emulator and server try to use them for building.

The *config* program is used to write the *Makefile.internal* file that directs the building of the Unix server just as it did for the kernel. Configurations are again specified by the files in the *ux/server/conf* directory. Because of license restrictions on the Unix server sources, you may not be able to build the default configuration that we build at CMU. This configuration, *STDAFS+WS*, requires sources for the kernel support for NFS and AFS. If you do not have source liceses for these, you will need to change the *CONFIG* variable to *STDVICE+WS*. *CONFIG* is defined in *ux/Makeconf*. It can either be changed there or defined on the command line to *odemake*.

5.4. Configurations

Both the mach kernel and Unix server use a modified version of the BSD *config* program to write *Makefile.internal* which is used to build the kernel/server. There are three types of files that provide input to *config*: *MASTER*, *template.mk* and *files*.

The *MASTER* files define the various individual configuration options and a few macro configuration names that are combinations of individual options. For each machine type the options from *MASTER, MASTER.local, <traget_cpu>/MASTER* and *<target_cpu>/MASTER.local* are combined by the *doconf* script into a file in the object directory with the name *conf/<CONFIG>*. Thus if you are building a *STD+WS* configuration of the kernel, the file *obj/@sys/mk/kernel/conf/STD+WS* will contain all the options selected for this configuration.

The macro-configuration definition lines are of the form:

```
#  STD       = [ debug mdebug ipc_debug vm_debug pm fixpri pcs ]
```

Note that this is not a comment as are most of the other lines that start with "#", but is read by the *doconf* script. You can add your own custom configuration in a similiar manner to the appropriate MASTER file.

The *files* files simply list each file, conditionalized by the options that will select it, that is to be used in the kernel or server. Again the *files* and *<target_cpu>/files* are combined. The lines *OPTIONS/foo* cause a file *foo.h* to be written into the object directory which contains a *#define FOO 1|0* depending on whether the option foo was selected or not.

The *template.mk* and *<target_cpu>/template.mk* contain definitions and template rules for *config* to use in writing *Makefile.internal*.

The *Makeconf* file defines the *CONFIG* variable which is used to chose the configuration to build. This can be overridden by setting it on the command line to odemake. For example the *odemake* command in the *ux* directory could be

```
odemake CONFIG=STDVICE+WS
```

which would cause the non-AFS/NFS configuration of the Unix server to be built. It is also possible to select or remove specific options in the following manner: for example in *mk*

```
odemake CONFIG=STD+WS-debug-ipc_debug
```

would disable the kernel debugging interfaces. Note: we do not expect that most randomly chosen configurations would either build or work. There are several drivers for various machines which are not usually built and are likely to require some work before they will compile or run.

5.5. Special files

A release area contains several primary subdirectories. Most of these, *bin, lib, include, etc, man* should be familiar to you. The *special* subdirectory is where the following binaries are installed:

```
mach.boot.<version>.<config>
emulator.<version>
vmunix.<version>.<config>
```

where *<version>* is a string like *MK69* or *UX31*, and *<config>* is a string like *STD+ANY* or *STDAFS+WS*.

These files should be copied from the special directory when installing them in */* or */mach_servers*. The files in the object directory are NOT stripped. For all kernels MK68 and later, you should install and boot *mach.boot*, NOT *mach_kernel*. *mach.boot* is produced from *bootstrap* and *mach_kernel* with the *makeboot* program. What name you give *mach.boot* in / or /../.. is of course a matter for you and your boot loader to decide. For more detailed information on installing these programs see the document **Setup for Mach 3.0**.

6. Shadowing

The Makefiles also support shadowing. With shadowing, you can have a "system source area" which contains most source files, and a "local source area" which only contains files that you have changed. In the extreme, the only file you need in your local source area is the top-level *Makeconf*. (Each collection contains one *Makeconf* now, at top-level.) A shadow build searches your local source area for files and directories first, and then checks the system source area for files and directories. To do shadowing, you should call setvar.csh with -masterbase option. For example, if your local area is */usr/rvb/mymach3/src/mk* and the system area is */afs/cs/project/mach3/latest/src/mk* then:

```
cd /usr/rvb/mymach3/src/mk/bootstrap
setvar -masterbase /afs/cs/project/mach3/latest i386_mach
odemake
```

does a shadow build of *bootstrap* in the *mk* collection. Any libraries or include files that are not in your local release area will be found in the system release area, */afs/cs/project/mach3/release/latest/@sys*. The object, export and release files you build will be placed respectivley in */usr/rvb/mymach3/{obj,export,release}/@sys*.

7. Pointers to related topics

7.1. SUPING

If you are not at CMU the standard way to get both the sources and the system release area is to use our Software Upgrade Protocol program (SUP) to copy the files over the Internet. There is a different collection for each source subtree and one for the system release area. Different licenses are required to get permission to SUP the different collections. Your machine must be enabled in our host tables to SUP these collections. Information on all the details can be found in */usr/mach/public/doc/notes/mach3_supinfo* which can be FTP'ed from mach.cs.cmu.edu as user *anonymous*.

7.2. m3tree

If you are at CMU and have read access to the mach3 archive area, you can get a copy of the sources by using the *m3tree* script. Use of this script is documented by calling it with the -help option and in */afs/cs/project/mach/public/doc/notes/mach3_cmu.{ps.doc}*.

7.3. How to run Mach 3.0

See the document **Setup for Mach 3.0** which can be found in */usr/mach/public/doc/notes/mach3_setup.{ps,doc}*

Table of Contents