Building Mach 3.0

Mary R.Thompson
Richard P. Draves
21 May 1992

1. Expected Environment

The Mach 3.0 build process assumes that it is being performed on a system that is running either Mach 3.0, Mach 2.5, MSD 2.6 or OSF/1. Otherwise many of the tools used in the build process may not be consistent with the ones that are expected. We provide sources for some, but not all, of our build tools.

The biggest difference between our tools and standard BSD Unix is our use of searchpaths. Our *cpp* looks for include files along the environment variable *CPATH*. *make* has an additional searchpath, *MAKECPP*, that it hands to *cpp*. To help alleviate this difficulty, we have included binaries for *make*, *cc* and *cpp* or *gcc-cpp* in the release area. This is a collection that is suppable by holders of BSD 4.3 licenses. If you have supped this collection install the programs in it at the beginning of your *PATH*. The man page for *make* is also included in our *mach3.release* area.

2. Directory Structure

The Makefiles make some assumptions about the layout of source, object and release areas. In order to avoid modifications to the Makefiles, you should have your sources in adjacent directories with the following names: ***mk*** (kernel sources), ***ux*** (Unix-server sources), ***user*** (user level program) and ***poe*** (the POE server). The object area (where the .o files and executables are built) and the release area (where includes, libraries and programs are executed from) should be adjacent to the src directory.

The source, object and release directory organization is now:

```
src/{mk,ux,user,...}/...
obj/{pmax_mach,i386_mach,...}/{mk,ux,user,...}/...
release/{pmax_mach,i386_mach,...}/{bin,lib,include,etc,man,special}/...
```

Each collection has is its own source area and object area (for example, *src/mk* and *obj/@sys/mk*) but the collections share a release area, *release/@sys*. You must set up the source directories and top level object and release directories and populate them either by ***sup*** or ***m3tree***. The build process creates the obj and release subdirectories. Having objects placed in a machine dependent directory that is separate from the source directory allows us to build objects for several machine types from the same sources in the same logical name space. On filesystems such as NFS and AFS, it also allows the source directories to be "read-only" and the object directories to be treated as more "temporary" files.

The CMU @***sys*** variable (which on Mach systems expands to <*machine*>*_mach*) is used in the release and install directories. It appears in the *Makeconf* files where it can be overridden by defining the *make* variable ***ATSYS***. It also appears in the *Makefile-mklinks* files.

3. Overview of Build Process

Each component builds using files from the release area and installs files back into the release area. This means that if you are building several components, then they must be built in the proper order. Bootstrapping from scratch, e.g. when the release area starts off empty, is a little tricky, and we supply a script that can be used as a template. However, if you have a "system release area" that is already populated, then it can be searched for files after the "local release area". If you are working on a platform that is supported by CMU, you can get the contents of this release area from the *mach3.release* SUP collection. In this case, you only have to build the components you are interested in, and everything else comes from the system release area. For example, you no longer have to build *mig, config, makeboot, libraries, bootstrap*, etc. to build a kernel. You no longer have to build a kernel before building a Unix server.

The Makefiles (Makeconfs, actually) take care of adding the local release area to your *PATH/CPATH/LPATH* variables so that it is searched appropriately. If you are not bootstrapping from scratch, you should add the system release area to your variables yourself. For example,

```
setpath -i0 /usr/mach3
```

if */usr/mach3* is your system release area. The system release area at CMU is

```
/afs/cs.cmu.edu/project/mach3/latest/release/@sys
```

.

The Makefiles use the *wh* program to search path variables. For example, dependencies on libraries are generated by searching *LPATH* for the library. The user collection includes source for the *wh* program and it is installed in the system release area. The source for *md* and *xstrip* programs used by the Makefiles is also in the user collection.

The top-level Makefiles *(mk/Makefile, ux/Makefile, user/Makefile*) build the subdirectories in the collection in the appropriate order. Only subdirectories which actually exist in your copy of the sources are built. For example, if you have *mk/bootstrap/** and *mk/kernel/** but not *mk/include/** and not *mk/user/**, then builds in *mk* will skip the non-existent directories. Instead, include files and libraries will come from the system release area. However, the top-level Makefile will build *mig, config, makeboot, doconf* (from *mk/kernel/src/**) and install them in the local release area before building the kernel so that they can be used from there for the kernel build. Also machine dependent directories for platforms other than the one being built may be missing.

This document does not try to duplicate all of the information in the CS *make* man page. It is expected that you read it along with this document. Some of the variables that our build process uses are used and/or defined and/or modified by the *make* program. Others are purely local, defined and used only by our Mach 3.0 Makefiles. It is important to understand which are used by *make* before attempting any changes to the Makefiles.

One feature of CMU's *make* that you want to avoid is its default behavior of always looking for an RCS archive file, e.g. foo.c,v and RCS/foo.c,v, when it fails to find a primary file, foo.c. If you keep all your source files checked out, and have several directories that are being searched for prerequisite files, this can add a noticeable and unnecessary amount of time to a build, especially one where most of the time is spent on *fstat*'s of files that are up-to-date. To avoid this behavior run *make* with the **-c** switch. For those people who are doing builds from a shadow area that has *RCS* links back to the Mach 3.0 archive areas, *make* may actually check-out obsolete versions of files from the RCS tree instead of finding a newer version that may now be in a different directory.

4. Makefiles

4.1. targets

The Makefiles have four main targets. The default target is *install*. That is, *make* is equivalent to '*make install*'.

`make install`	builds and installs into the local release area
`make clean`	removes files installed in the local release area
`make all`	When possible, builds in the object directory without installing.
`make release`	Used for copying from the local release area to a system release area, or copying between system release areas.

The toplevel Makefiles also have a target ***mkidirs*** to create the primary subdirectories in the local release directory. Normally the Makefiles do not assume that the object area or the local release area have been populated with subdirectories. Instead, subdirectories are created as needed. This means before building, only the top-level object directory *(obj/@sys/<collection>*) and release directory *(release/@sys*) must already exist. If you are not going to build from the top, but are going to go straight into a subdirectory and build, to ensure that *bin, lib, etc* are created first before you try to install into them, you should make the target ***mkidirs*** at the top level.

Similarly, the target ***mkrdirs*** creates the primary subdirectories in the release directory, which you should do before going straight into a subdirectory and doing a '*make release*'. A '*make install*' at top-level automatically does a '*make mkidirs*', and a '*make release*' at top-level automatically does a '*make mkrdirs*', so normally you shouldn't have to worry about this.

4.2. Generic Makefiles

Each collection has its own set of standard Makefiles which are included by the individual Makefiles in the subdirectories. ***Makefile-common*** contains standard definitions and few very general rules. It is included by all the individual Makefiles. The other ***Makefile-**** files at the top level define more specialized rules and are only included by Makefiles that need those rules.

The ***Makeconf*** file at the top level is read by make before it reads any other Makefile. *Makeconf* is used to set the *make* variables that determine where the objects are to be built and the binaries are to be installed. It also sets the variables that determine what directories are searched for executables, libraries and include files. Make starts in the directory that it was called in and searches up the directory hierarchy to find a *Makeconf* file. It uses the first one it finds. If there is no *Makeconf* file anywhere in the superior tree then none is used.

4.3. Make variables

CONFIG
: Selects what configuration options are chosen for the kernel and Unix server. Defined in *Makefile-config*.

OBJECTDIR
: The directory in which the objects are placed. If this is undefined the objects are put in the source directory. Defined in *Makeconf*.

INSTALLDIR
: The directory in which executable files, include files and libraries are put by the install target. Defined in *Makeconf*.

FRELEASEDIR
: The directory in which files are found by *make release*. Defaults to *INSTALLDIR*. Defined in *Makeconf*.

TRELEASEDIR
: The directory into which files are installed by *make release*. Defined in *Makeconf*.

atsys
: Used as the name of the machine-dependent directories in the object and release areas. Defined in *Makeconf*. Defaults to *@sys*, unless the environment variable *ATSYS* is set.

MAKECONF
: Defined by *make* 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 *make* as the relative pathname from the directory in which *make* was invoked to the directory containing the *Makeconf* file.

TARGET_MACHINE
: Defined by *make* as the machine type for which the *make* is taking place. It is used in our Makefiles to select definitions based on which machine type the build is for. It is also used by *make* when it is called with the -m switch as as additional subdirectory to search for prerequisite files.

VPATH
: The search path that *make* uses to find the prerequisite files when it is evaluating a dependency rule. This may be set by individual Makefiles, but *make* 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. Running *make* with the -d switch will show you what *make* is using for the *VPATH*

cpu
: Yet another name for machine dependent subdirectories. Defined in Makefile-common and can be used as a component of VPATH for directories which have machine-dependent subdirectories.

DIRS
: Defined by toplevel Makefiles as the list of subdirectories in which *make* will be run by the RECURSE template.

MKIDIRS
: Defined by toplevel Makefiles as the directories that must be created in the install area.

MKRDIRS
: Defined by the toplevel Makefile as the directories that must be created in the release area.

PROGRAM,OBJECTS,LIBRARIES
: Defined by leaf Makefiles. These variables are used in ways defined by the toplevel Makefile-* files. In general *PROGRAM* is the target being built, it depends on *OBJECTS* and *LIBRARIES* and it links with *LIBRARIES*. Details may vary depending on where in the tree the Makefile is and what Makefile-* it is including.

4.4. Environment variables

CPATH	The search path for include files. Defined in *Makeconf* and some Makefiles. Used by *cpp*.
LPATH	The search path for libraries. Defined in *Makeconf*. Used by ld and some Makefiles.
PATH	The search path for executables. Defined in *Makeconf*. Used by *csh*.
MAKECPP	A search path defined by make and passed to *cpp* for *cpp* to search for include files. You can check the value of this path by running *make* with the -d switch.
MAKECWD	Defined by *make* as the absolute path to the directory where *make* is currently executing; exported to the environment.
MAKEDIR	Defined by *make* as the relative path from the directory containing the *Makeconf* file to the directory in which *make* was invoked; exported to the environment.
MAKEPSD	Defined by *make* as a pathlist of the source directories. If you are not using any shadow directories, this is just the relative pathname back to the source directory from the object directory. If you are building from shadow sources, this will also include the master soruce directory; exported to the environment.
CPP_INCL_DEBUG	If set to 1, *cpp* prints out the directories that it is searching for include files.

5. Cross compiling

With the release of MK73, UX34, and USER16, the Makefiles support cross-compilation. Johannes Helander of the Helsinki Institute of Technology did the original development, and then Rich Draves incorporated it into our current Makefiles. Johannes also added a clean way for sites to override the standard make definitions.

If you aren't cross-compiling, the cross-compilation support shouldn't get in your way. Everything works as before.

In addition to the old *TARGET_MACHINE* variable (set by make), there is now a *HOST_MACHINE* variable, which defaults to *${TARGET_MACHINE}*. You cross-compile by overriding the default values of these variables so that their values are different.

When cross-compiling, some stuff (meant for the host machine) is installed in a different location. There is a *HOST_INSTALLDIR* variable which defines the location of the local host release area. It defaults to *${INSTALLDIR}/${host_atsys}*. For example, if one is compiling for the i860 on i386 *(TARGET_MACHINE=I860, HOST_MACHINE=I386*) then the directory layout is

```
source          src/mk
objects         obj/i860_mach/mk
target release  release/i860_mach/{bin,lib,...}
host release    release/i860_mach/i386_mach/{bin,lib,...}
```

Programs like *mig, makeboot, config, md, wh, xstrip* are compiled for the host machine and installed in the host release area. The path variables *PATH, HOST_LPATH*, and *HOST_CPATH* search the host release area. The path variables *LPATH* and *CPATH* search the target release area.

For 'make release', there are variables *HOST_FRELEASEDIR* and *HOST_TRELEASEDIR* which work the way you'd expect.

If you aren't cross-compiling, then *HOST_INSTALLDIR* = *INSTALLDIR*, etc, and this all collapses to the normal directory layout.

6. Overriding the Makefile defaults

To override the standard definitions in Makeconf, you can create a **Makeconf-site**, which *Makeconf* includes if it exists. In *Makeconf-site* you can define *SOURCEDIR* (to get shadowing), redefine *OBJECTDIR* & *INSTALLDIR*, override *HOST_MACHINE* & *TARGET_MACHINE*, define initial values for the path variables, etc.

To override the standard definitions in *Makefile-common*, you can create a **Makefile-site**, which *Makefile-common* includes it it exists. In *Makefile-site* you can add/override definitions of *CC* and the like.

7. Building the *mk* collection

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 *make* 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.

We have recently added the 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 only the Makefiles to install the *mk* collection include files.

`user/libmach` — Makefile and sources to build 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 *make* with the newly created *Makefile.internal* as the Makefile that actually compiles files for the kernel.

When you do a *make* in the *mk* directory, it will proceed to make the install target in each of the subdirectories in the ***DIRS*** list in *kernel/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 installed. One item to notice, is that the *kernel/Makefile* resets *CPATH* to ".", before doing the *make* with *Makefile.internal*. This has the effect of having the kernel *make* look for include files only within the *kernel* source and object directories, rather than from the release area.

8. 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**, *Makefile** 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, MASTER.<machine>* and *MASTER.<machine>.local* are combined by the *doconf* script into a file in the object directory with the name *<CONFIG>*. Thus if you are building a *STD+WS* configuration of the kernel, the file *obj/@sys/mk/kernel/STD+WS/STD+WS* will contain all the options selected for this configuration.

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 *files.<machine>* 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 ***Makefile.template*** and ***Makefile.<machine>*** contain definitions and template rules for *config* to use in writing *Makefile.internal*.

The *Makefile-config* 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 make. For example the *make* command in the *ux* directory could be

```
make 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*

```
make 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.

9. 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/Makefile-config*. It can either be changed there or defined on the command line to *make*.

10. 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:

```
bootstrap.<version>
mach_kernel.<version>.<config>
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**.

11. Bootstrapping

We are still not at the point where we can provide a full set of bootstrap files and procedures. The main things that are missing are the CMU version of *make* and a CMU compatible version of *cpp*. We do make our modifications to *gcc-cpp* available in the */usr/mach/public/src* ftp area. The *make* we use is based on BSD 4.3 and is encumbered with AT&T licenses. We hope to change to a non-licensed *make* at some point.

Meanwhile Rich Draves has provided a shell script called ***m3build*** which will build and install the tools that we do supply in the correct order. It can be found in */usr/mach/public/doc/mach3/m3build.sh*. It should be run from the *<base>/src* directory. It is run as:

```
m3build  bootstrap | mk | ux | user
         [-mach3release <release-dir>] [-clean] [-extra]
```

'*m3build bootstrap*' builds and installs *md, wh, xstrip*, the *mk* and *ux* include files and *libcmucs*. '*m3build mk* **|** *ux* **|***user*' builds and installs all files in those trees. The *-mach3release* switch lets you specify the 'system release area' where files like *make* and *cc* or *gcc* are found. The *-clean* switch will delete any of previously released files or objects before it does the build. *-extra* will build all the configurations of *mk* or *ux*.

12. 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 set ***SOURCEDIR*** to the system source and build in your local source area. For example, if your local area is */usr/rpd/mymach3/src/mk* and the system area is */afs/cs/project/mach3/latest/src/mk* then:

```
cd /usr/rpd/mymach3/src/mk/bootstrap
setpath -i0 /afs/cs/project/mach3/release/latest/@sys
make SOURCEDIR=/afs/cs/project/mach3/latest/src/mk install
```

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*. Your copy of *Makeconf* in */usr/rpd/mymach3/src/mk* should define *OBJECTDIR* and *INSTALLDIR* to be someplace appropriate for your files to be placed in. You can also define *SOURCEDIR* in *Makeconf*.

13. Other Topics

13.1. Supping

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/mach3/mach3_sup*.

13.2. 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 /usr/mach/public/doc/mach3/mach3_cmu.{ps.doc}.

13.3. How to run Mach 3.0

See the document **Setup for Mach 3.0** which can be found in */usr/mach/public/doc/mach3/mach3_setup.{ps,doc}*

Table of Contents