Fiasco's build system

Michael Hohmuth

February 23, 2004

Contents

1 Motivation

My main goal when I designed Fiasco's build system was to allow for multiple configurations of the kernel, based on different implementations of the same interface, to coexist in multiple object directories.

2 Configuration

Fiasco's build system consists of a number of configuration files, and it also makes use of the L4 source tree's global Make configuration file, l4/Makeconf. All configuration files are written in Makefile language.

l4/Makeconf Global Make rules and search paths

l4/kernel/fiasco/src/Makefile "Fiasco's build system" rules

l4/kernel/fiasco/src/Modules.in Standard configuration for Fiasco

<object directory> /Modules Current configuration for Fiasco

l4/kernel/fiasco/src/Makerules.* Make rules for each subsystem defined in Modules

l4/kernel/fiasco/src/Makerules.local' (Optional) User-specific configuration files

l4/Makeconf.local' (Optional) User-specific configuration files

(By default, the object directory is the main source directory, l4/kernel/fiasco/src/. See the next section on how to use other object directories with custom configurations.)

Users configure the build system by creating a Modules file. If no Modules file exists, the standard configuration Modules.in is copied to Modules. The build system warns the user if (after a "cvs update") Modules.in is newer than Modules.

The Modules file defines a number of subsystems that should be built. For each subsystem, there must be a Makerules.<subsystem> file that defines rules for building the subsystem's targets.

In the remainder of this section, I describe by example the language used in Modules file and the contents of the Makerules.* files.

```
SUBSYSTEMS = FOO BAR
       # Defines two subsystems, FOO and BAR.  This means that there
       # exist two files, Makerules.FOO and Makerules.BAR, that
       # contain rules on how to build the targets of these
       # subsystems.  These targets are defined later.

### Definitions for subsystem FOO follow

FOO = foo
       # Defines the main target of subsystem FOO: a file named
```

```
        # "foo".

FOO_EXTRA = foo.man
        # (Optional) Defines more targets that should be built for
        # subsystem FOO.

INTERFACES_FOO = foo1 foo2 foo3
        # (Optional) C++ modules for subsystem FOO (written in
        # ‘preprocess’ format; see
        # <URL:http://os.inf.tu-dresden.de/~hohmuth/prj/preprocess/>)
        # Each module normally consists of one implementation file
        # such as foo1.cpp -- unless IMPL definitions such as the
        # following ones are given:

foo2_IMPL = foo2 foo2-more
        # (Optional) C++ module foo2 is implemented in two files
        # foo2.cpp and foo2-more.cpp (instead of just foo2.cpp).  The
        # public header file generated from these implementation files
        # will be called foo2.h.

foo3_IMPL = foo3-debug
        # (Optional) C++ module foo3 is implemented in foo3-debug.cpp,
        # not foo3.cpp.  The public header file generated from this
        # implementation file will be called foo3.h.

CXXSRC_FOO = frob1.cc frob2.cc
        # (Optional) Additional C++ sources for subsystem FOO (not in
        # ‘preprocess’ format)

CSRC_FOO = frob3.c frob4.c
        # (Optional) Additional C sources for subsystem FOO

ASSRC_FOO = frob5.S
        # (Optional) Additional assembly-language sources for
        # subsystem FOO

OBJ_FOO = frob6.o
        # (Optional) Additional objects for subsystem FOO.  These
        # objects can be precompiled or generated using custom rules
        # in Makerules.FOO.

NOPROFILE += frob2
        # (Optional) Basenames of objects that should not be compiled
        # with profiling options in profiling builds.

NOOPT += frob3
```

```
        # (Optional) Basenames of objects that should not be compiled
        # with optimization options.

PRIVATE_INCDIR += incdir
        # (Optional) Add incdir to the include path for all source
        # files.  (This feature is implemented by l4/Makeconf.)

VPATH += foodir
        # (Optional) Add foodir to Make's source-file search
        # path.  (This feature is implemented internally by Make.)

### Definitions for subsystem BAR follow
### (similar to FOO's definitions)
```

The Makerules.FOO file usually contains just rules for linking the subsystem's targets. Additionally, it must contain a rule "clean-FOO:that cleans the object directory from files created by this configuration file.

It can access the following Make variables:

`FOO`, `FOO_EXTRA` names of targets

`OBJ_FOO` expanded to contain *all* objects that will be created for subsystem FOO

`BAR` targets of other subsystems

3 Building in separate object directories

It is possible to configure multiple directories, each with its own Modules file, as separate object directories. (This usage is supported only if the main source directory [the one containing Modules.in] is not also used as an object directory.)

To use a directory as an object directory, create in it a Makefile like this:

```
srcdir = ..

all:

%:
          $(MAKE) -I $(srcdir) -f $(srcdir)/Makefile \
                  srcdir=$(srcdir) $@
```

Change the "srcdir" definition to point to the main source directory. You can then create custom Modules files (and custom source files and Makerules.* files) in each object directory.

References