# RTEMS Frequently Asked Questions

Edition 4.6.1, for 4.6.1

5 March 2004

**On-Line Applications Research Corporation**

On-Line Applications Research Corporation
TEXinfo 2002-11-25.11



The RTEMS Project is hosted at http://www.rtems.com. Any inquiries concerning RTEMS, its related support components, its documentation, or any custom services for RTEMS should be directed to the contacts listed on that site. A current list of RTEMS Support Providers is at http://www.rtems.com/support.html.

# Table of Contents

# 1 Basic Information

The questions in this category are basic questions about RTEMS. Where did it come from, why is it, what is it, when should you use it, etc.?

## 1.1 What does RTEMS stand for?

RTEMS is an an acronym for the Real-Time Executive for Multiprocessor Systems.

Initially RTEMS stood for the Real-Time Executive for Missile Systems but as it became clear that the application domains that could use RTEMS extended far beyond missiles, the "M" changed to mean Military. At one point, there were both Ada and C implementations of RTEMS. The C version changed the "M" to mean Multiprocessor while the Ada version remained with the "M" meaning Military.

## 1.2 What is RTEMS?

RTEMS is a real-time executive which provides a high performance environment for embedded military applications including many features. The following is just a short list of the features available in RTEMS. If you are interested in something that is not on this list, please contact the RTEMS Team. Features are being added continuously.

- Standards Compliant
    - POSIX 1003.1b API including threads
    - RTEID/ORKID based Classic API
- TCP/IP Stack
    - high performance port of FreeBSD TCP/IP stack
    - UDP, TCP
    - ICMP, DHCP, RARP
    - TFTP
    - RPC
    - FTPD
    - HTTPD
    - CORBA
- Debugging
    - GNU debugger (gdb)
    - DDD GUI interface to GDB
    - thread aware
    - debug over Ethernet
    - debug over Serial Port
- Filesystem Support

    - In-Memory Filesystem (IMFS)
    - TFTP Client Filesystem
- Basic Kernel Features
    - multitasking capabilities
    - homogeneous and heterogeneous multiprocessor systems
    - event-driven, priority-based, preemptive scheduling
    - optional rate monotonic scheduling
    - intertask communication and synchronization
    - priority inheritance
    - responsive interrupt management
    - dynamic memory allocation
    - high level of user configurability

## 1.3 What standards does RTEMS support?

The original "Classic" RTEMS API is based on the Real-Time Executive Interface Definition (RTEID) and the Open Real-Time Kernel Interface Definition (ORKID). RTEMS also includes support for POSIX threads and real-time extensions.

With the addition of file system infrastructure, RTEMS supports about 70% of the POSIX 1003.1b-1996 standard. This standard defines the programming interfaces of standard UNIX. This means that much source code that works on UNIX, also works on RTEMS.

## 1.4 What processors is RTEMS available for?

RTEMS is available for the following processor families:

- Motorola MC68xxx
- Motorola MC683xx
- Motorola ColdFire
- ARM
- Hitachi H8/300
- Hitachi SH
- Intel i386
- Intel i960
- MIPS
- PowerPC
- SPARC
- AMD A29K
- Hewlett-Packard PA-RISC
- Texas Instruments C3x/C4x

- OpenCores OR32

In addition, there is a port to UNIX which can be used as a prototyping and simulation environment.

## 1.5 Executive vs. Kernel vs. Operating System (RTOS)

The developers of RTEMS developers use the terms executive and kernel interchangeably. In the embedded system community, the terms executive or kernel are generally used to refer to small operating systems. So we consider it proper to refer to RTEMS as an executive, a kernel, or an operating system.

## 1.6 Where/why was it developed?

RTEMS was developed by On-Line Applications Research Corporation (OAR) for the U.S. Army Missile Command prior to that organizations merger with the Aviation Command that resulted in the new command, U. S. Army Aviation and Missile command (AMCOM). The original goal of RTEMS was to provide a portable, standards-based real-time executive for which source code was available and royalties were paid.

In other words, RTEMS was open source before open source was cool.

Since the initial release to the world, the RTEMS Community has grown enormously and contributed significantly to RTEMS. Important additions such as the TCP/IP stack, FAT filesystem, multiple ports, device drivers, and most BSPs have come from users like yourself.

## 1.7 Are there no similar commercial products?

Yes, but not all are based on standards and the open source philosophy.

## 1.8 How can I get RTEMS?

RTEMS is distributed from http://www.rtems.com. This is a server dedicated to the RTEMS Project which was donated by and hosted by OAR Corporation to provide a focal point for all RTEMS activities. Point your favorite browser at the following URL and following the link:

http://www.rtems.com

But if you are already reading this, you probably already found it. :)

## 1.9 What about support?

RTEMS development and support services are available from a number of firms. See http://www.rtems.com/support.html for the current list of RTEMS service providers.

Remember that RTEMS maintenance is funded by users. If you are using RTEMS on a commercial project, please get support.

## 1.10 Are there any mailing lists?

The primary RTEMS mailing list is `rtems-users@rtems.com`. This list is for general RTEMS discussions, questions, design help, advice, etc.. Subscribe by sending an empty mail message to `rtems-users-subscribe@rtems.com`. This mailing list is archived at:

`http://www.rtems.com/ml/rtems-users`

The `rtems-snapshots@rtems.com` mailing list is for those interested in taking a more active role in the design, development, and maintenance of RTEMS. Discussions on this list tend to focus on problems in the development source, design of new features, problem reports, etc.. Subscribe by sending an empty mail message to `rtems-snapshots-subscribe@rtems.com`. mailing list is archived at:

`http://www.rtems.com/ml/rtems-snapshots`

The archives for both mailing lists include discussions back to 1997.

## 1.11 Are there any license restrictions?

RTEMS is licensed under a modified version of the GNU General Public License (GPL). The modification places no restrictions on the applications which use RTEMS but protects the interests of those who work on RTEMS.

The TCP/IP network stack included with RTEMS is a port of the FreeBSD network stack and is licensed under different terms that also do not place restrictions on the application.

## 1.12 Are there any export restrictions?

No.

# 2 General Development Tool Hints

The questions in this category are related to the GNU development tools in a non-language specific way.

## 2.1 What is GNU?

Take a look at http://www.gnu.org for information on the GNU Project.

## 2.2 How do I generate a patch?

The RTEMS patches to the development tools are generated using a command like this

```
diff -N -P -r -c TOOL-original-image TOOL-with-changes >PATCHFILE
```

where the options are:

- -N and -P take care of adding and removing files (be careful not to include junk files like file.mybackup)
- -r tells diff to recurse through subdirectories
- -c is a context diff (easy to read for humans)

Please look at the generated PATCHFILE and make sure it does not contain anything you did not intend to send to the maintainers. It is easy to accidentally leave a backup file in the modified source tree or have a spurious change that should not be in the PATCHFILE.

If you end up with the entire contents of a file in the patch and can't figure out why, you may have different CR/LF scheme in the two source files. The GNU open-source packages usually have UNIX style CR/LF. If you edit on a Windows platform, the line terminators may have been transformed by the editor into Windows style.

## 2.3 How do I apply a patch?

Patches generated with the `diff` program are fed into the `patch` program as follows:

```
patch -p1 <PATCHFILE
```

where the options are:

- -pNUM tells `patch` to strip off NUM slashes from the pathname.

If `patch` prompts for a file to patch, you may need to adjust NUM.

# 3 RTEMS Concepts

The questions in this category are hints that help basic understanding.

## 3.1 RTEMS Workspace versus Program Heap

The RTEMS Workspace is used to allocate space for objects created by RTEMS such as tasks, semaphores, message queues, etc.. It is primarily used during system initialization although task stacks and message buffer areas are also allocated from here. Section 6.3 [How do I determine how much memory is left?], page 20.

# 4 Building RTEMS

Building any package in a cross-compilation fashion can be difficult, but configuring and building a real-time operating system that supports many CPU families and target boards can be confusing. The RTEMS development team has made every effort to make this process as straight forward as possible but there are going to be questions.

Moreover, between RTEMS 4.0 and 4.5, the configure and Makefile system in RTEMS was changed to be more compatible with GNU standards. This transition has lead to a number of subtle differences.

This section of the FAQ tries to address the more frequently asked questions about building RTEMS. Thanks to Ralf Corsepius for compiling this section from excerpts from various postings to the rtems-users mailing list.

## 4.1 Required Tools

### 4.1.1 Which tools are required to build RTEMS?

- A native C-Toolchain, gcc prefered
- GNU-Bash and shell utils
- GNU-make.
- A target (typically cross) C- and (optional) C++-Toolchain.
- autoconf/automake (optional, recommended)
- Perl (optional, recommended, needed by automake and some tools within RTEMS)
- m4 (optional, recommended, needed by autoconf, GNU-m4 preferred)

### 4.1.2 Do I need autoconf and automake to build RTEMS?

No, you don't. Or to be more accurate, you won't need them until you modify something in RTEMS's Makefile.ams/configure.acs or start to develop with RTEMS.

I.e. you won't need them to get started, but you will need them when getting serious.

### 4.1.3 Do I need a native gcc on my host?

No, you should be able to use any native, ansi-compliant C-compiler, but using a native gcc is highly recommended.

### 4.1.4 Can I use a non-gcc cross-toolchain?

Generally speaking, it should be possible. However, most RTEMS development has taken place using gcc, therefore getting it working may not be easy.

### 4.1.5 Do I need gcc-2.9x for cross compilation?

[FIXME: Partially obsolete]

Not necessarily, but gcc-2.9x is highly recommended, because most development has taken place using gcc-2.9x and previous versions of gcc are not actively supported in RTEMS anymore (Section 4.1.4 [Can I use a non-gcc cross-toolchain?], page 9).

### 4.1.6 Where to get autoconf automake ld gcc etc.?

The sources of all gnutools are available at any GNU mirror. Native Linux binaries should come with any Linux distribution. Native Cygwin binaries should be available at Cygnus.

GNU-Toolchain binaries (gcc, binutils etc.) for Linux and patches required to build them from source are available from the RTEMS ftp site.

## 4.2 Issues when building RTEMS

### 4.2.1 When running ./configure weird thing start to happen

You are probably trying to build within the source-tree. RTEMS requires a separate build directory. I.e. if the sources are located at `/usr/local/src/rtems-4.6.1`, use something similar to this to configure RTEMS:

```
cd somewhere
mkdir build
cd build
/usr/local/src/rtems-4.6.1/configure [options]
```

### 4.2.2 When running bootstrap weird thing start to happen

Many possibile causes: Most likely one of these:

- You are trying to build RTEMS with insufficient or incompatible versions of autoconf and automake.
- The autotools can't be found because your $PATH might not be set up correctly (Cf. Section 4.2.7 [How to set up $PATH?], page 11)
- You have used configure-script options which interfer with RTEMS configuration (Cf. Section 4.4.10 [configure –program-[prefix|suffix|transform-name]], page 16)
- You have tripped over a bug in RTEMS ;)

### 4.2.3 configure xxx cannot create executables

While running a configure script, you see a message like this:

```
checking for m68k-rtems-gcc... (cached) m68k-rtems-gcc
checking for C compiler default output... configure: error: C compiler cannot create e
configure: error: /bin/sh '../../../../rtems-ss-4.6.1/c/make/configure'
failed for c/make
```

This kind of error message typically indicates a broken toolchain, broken toolchain installation or broken user environment.

Examinating the `config.log` corresponding to the the failing configure script should provide further information of what actually goes wrong (In the example above: `<target>/c/<BSP>/make/config.log`)

### 4.2.4 Why can I not build RTEMS inside of the source tree?

The build-directory hierarchy is setup dynamically at configuration time.

Configuring inside of the source tree would prevent being able to configure for multiple targets simultaneously.

Using a separate build-tree simplifies Makefiles and configure scripts significantly.

Adaptation to GNU/Cygnus conventions.

### 4.2.5 Which environment variables to set?

None. Unlike for previous releases, it is not recommended anymore to set any RTEMS related environment variable (Exception: $PATH, cf. Section 4.2.7 [How to set up $PATH?], page 11).

### 4.2.6 Compiler /Assembler /Linker report errors

If you see a bunch of the error messages related to invalid instructions or similar, then probably your `$PATH` environment variable is not set up correctly (cf. Section 4.2.7 [How to set up $PATH?], page 11). Otherwise you might have found a bug either in RTEMS or parts of the toolchain.

### 4.2.7 How to set up $PATH?

All target tools are supposed to be prefixed with a target-canonicalization prefix, eg. i386-rtems-gcc, m68k-rtems-ld are target tools.

Host tools are supposed not to be prefixed. e.g.: cc, ld, gcc, autoconf, automake, aclocal etc.

If using the pre-built tool binaries provided by the RTEMS project, simply prepend `/opt/rtems-4.6` to `$PATH`.

### 4.2.8 Can I build RTEMS Canadian Cross?

RTEMS >= 4.6.0 configuration is prepared for building RTEMS Canadian Cross, however building RTEMS Canadian Cross is known to be in its infancy, so your mileage may vary (See `README.cdn-X` in the toplevel directory of RTEMS's source tree for details.)

### 4.2.9 Building RTEMS is slow

RTEMS has become fairly large :).

In comparison to building previous versions, building RTEMS is slow, but that's the tradeoff to pay for simplier and safer configuration.

If using Cygwin, remember that Cygwin is emulating one OS ontop of another – this necessarily must be significantly slower than using U*nix on the same hardware.

### 4.2.10 Building my pre-4.5.x BSPs does not work anymore

See Section 4.4.1 [How to merge pre-RTEMS-4.5.0 BSPs into RTEMS-4.5.0?], page 14.

### 4.2.11 make debug_install / make profile_install

[FIXME:Partially obsolete]

These make targets are not supported anymore. Instead, use:

```
make VARIANT=DEBUG install
make VARIANT=PROFILE install
```

### 4.2.12 make debug / make profile

[FIXME:Partially obsolete]

These make targets are not supported anymore. Instead, use:

```
make VARIANT=DEBUG all
make VARIANT=PROFILE all
```

### 4.2.13 Building RTEMS does not honor XXX_FOR_TARGET

RTEMS < 4.6.0 did not support passing flags from the environment. If using RTEMS < 4.6.0, editing your BSP's `make/custom/mybsp.cfg` and setting appropriate flags there is required.

RTEMS >= 4.6.0 honors several XXX_FOR_TARGET environment variables. Run `<path-to-rtems>/configure --help` for a full list of supported variables.

### 4.2.14 Editing Makefile.in Makefile configure

These files are generated by auto* tools, cf. Section 4.2.15 [Editing auto* generated files], page 13).

### 4.2.15 Editing auto* generated files

RTEMS uses automake, therefore **never**, **ever**, **ever** edit Makefile.ins, Makefiles, configure or other auto* generated files. Changes to them will be swapped away soon and will get lost.

Instead edit the sources (eg.: Makefile.ams, configure.acs) auto* generated files are generated from directly.

If sending patches always send Makefile.ams and configure.acs. Sending Makefile.ins, Makefiles and configure scripts is pretty much useless. If sending larger patches, consider removing all auto* generated files by running `bootstrap -c` (cf. See Section 4.4.8 [./bootstrap], page 15) before running diff to cut a patch.

If you don't understand what this is all about, try start getting familiar with auto* tools by reading autoconf.info and automake.info, or feel free to ask for assistance on the RTEMS Mailing List (See Section 1.10 [Are there any mailing lists?], page 4.

## 4.3 Host Operating Systems and RTEMS

### 4.3.1 Can I use Windows or DOS?

No, plain DOS and plain Win will not work, but Cygwin should. Other U*nix emulations, such as Mingw and DJGPP are not supported and very likely will not work. Cywin / WinNT is known to work, but at the time of writing this, there seem to persist non-RTEMS related issues with Cygwin under Win9x which seem to prevent success on those systems.

### 4.3.2 Do I need Linux?

No, you should be able to build RTEMS on any U*ix OS and under Cygwin/NT (cf. Section 4.3.1 [Can I use Windows or DOS?], page 13).

### 4.3.3 Which Linux distribution is recommended?

None, any recent U*nix should work, i.e. any recent Linux distribution should work, too.

## 4.4 Development related questions

### 4.4.1 How to merge pre-RTEMS-4.5.0 BSPs into RTEMS-4.5.0?

[FIXME:Partially obsolete]

The simple answer is that between 4.0 and 4.5.0, RTEMS has moved to automake and greater compliance with GNU conventions. In 4.0, there was a single configure script at the top of the tree. Now RTEMS is configured more like other GNU tools – as a collection of configurable entities.

Each BSP now has its own configure script. I highly recommend you look at the Makefile.am's, configure.ac, of a similar BSP. You might even want to consider running "bootstrap -c" from the top of the tree and looking at what is left. bootstrap (cf. Section 4.4.8 [./bootstrap], page 15) generates/removes all automatically generated files.

### 4.4.2 What is no_bsp / no_cpu?

`no_bsp` is a fictional BSP for a fictional CPU of type `no_cpu`. `no_cpu/no_bsp` support files in RTEMS can be used as templates when implementing BSPs or porting RTEMS to new CPUs.

### 4.4.3 What is the bare-BSP?

At the time being RTEMS is build per BSP, with all support files being build separately for each BSP. This can become unhandy when using several similar but not identical boards (e.g. a PC with different peripherial cards plugged in), because this in general requires to implement a BSP for each setup. The bare BSP is a general, setup independent BSP which can be used when keeping all BSP specific parts external from RTEMS.

At present time the bare BSP is in its infancy. It is known that it can be build for most CPUs RTEMS supports. It is also known to work in individual cases, but your mileage may vary.

### 4.4.4 What is the cpukit?

[FIXME:To be extended]

One major change having been introduced to RTEMS-4.6.0 is the cpukit, located below the directory `cpukit/` in RTEMS's toplevel directory.

### 4.4.5 Multilib vs. RTEMS CPU-variants

The GNU toolchain applies a specific classification of similar CPUs into CPU variants (eg. SH1, SH2 etc.) to provide better support for each CPU variant.

RTEMS uses a different classification because it internally requires more details about a specific CPU than the GNU toolchain's multilib classification provides.

### 4.4.6 Keeping auto* generated files in CVS

When using CVS to archive source code, problems arise from keeping generated files in CVS. In general, two possible solutions exist:

- Find a way to get correct timestamps after checking out the sources from CVS. Some people try to achieve this by
  - carefully checking in files into CVS in appropriate order
  - applying scripts to fix timestamps accordingling (eg. by applying `touch` and `find`).
- Not keeping generated files in CVS, but regenerate them after having checked them out from CVS.

RTEMS favors the the latter variant, because it appears to be less error-prone and easier to handle (cf. Section 4.4.8 [./bootstrap], page 15 for details).

### 4.4.7 Importing RTEMS into CVS/RCS

When importing RTEMS into CVS/RCS or similar, we recommend not to import auto* generated files (cf. Section 4.4.6 [Keeping auto* generated files in CVS], page 15).

To remove them before importing, run

```
./bootstrap -c
```

from the toplevel directory of the source tree (cf. Section 4.4.8 [./bootstrap], page 15).

### 4.4.8 ./bootstrap

`bootstrap` is a simple shell script which automatically generates all auto* generated files within RTEMS's source tree (Other packages use the name `autogen.sh` for similar scripts). You will need to have autoconf, automake and further underlying packages installed to apply it.

It typically should be applied when having:

- checked out RTEMS sources from a CVS repository which does not contain generated files.
- added new automake / autoconf files to the source tree (eg. having added a new BSP), and when not being sure about what needs to be updated.

Once all autoconf/automake generated files are present, you will rarely need to run `bootstrap`, because automake automatically updates generated files when it detects some files need to be updated (Cf. Section 4.4.9 [configure –enable-maintainer-mode], page 16).

, ./bootstrap, Development related questions,]

### 4.4.9 configure –enable-maintainer-mode

When working within the source-tree, consider to append `--enable-maintainer-mode` to the options passed to configure RTEMS.

```
<path>/rtems-4.6.1/configure <options> --enable-maintainer-mode
```

This will enable the maintainer-mode in automake generated Makefiles, which will let automake take care about dependencies between auto* generated files. I.e. auto* generated files will get automatically updated.

Configuring RTEMS in maintainer-mode will require to have autoconf, automake and underlying tools installed (Cf. Section 4.1 [Required Tools], page 9).

### 4.4.10 configure –program-[prefix|suffix|transform-name]

These are generic configure script options automatically added by autoconf. RTEMS configuration does not support these, worse, they interfer with RTEMS's configuration – i.e. **do not use them**.

, Development related questions,]

### 4.4.11 configure.ac vs. configure.in

autoconf < 2.50 used the name `configure.in` for it's input files. autoconf >= 2.50 recommends using the name `configure.ac`, instead.

RTEMS > 4.5.0 applies autoconf >= 2.50, therefore all former RTEMS's `configure.in`'s have been renamed into `configure.ac` and have been adapted to autoconf >= 2.50 demands.

### 4.4.12 Reporting bugs

Several possibilities (In decreasing preference):

- File a bug report at RTEMS's GNATS
- Send an email to rtems-bugs@rtems.com
- Report your problem to one of the RTEMS mailing lists (Cf. Section 1.10 [Are there any mailing lists?], page 4).

# 5 BSP Questions

The items in this category provide answers to questions commonly asked about BSPs.

## 5.1 What is a BSP?

BSP is an acronym for Board Support Package.

A BSP is a collection of device drivers, startup code, linker scripts, and compiler support files (specs files) that tailor RTEMS for a particular target hardware environment.

## 5.2 What has to be in a BSP?

The basic set of items is the linker script, bsp_specs, and startup code. If you want standard IO, then you need a console driver. This is needed to run any of the RTEMS tests. If you want to measure passage of time, you need a clock tick driver. This driver is needed for all RTEMS tests EXCEPT hello world and the timing tests. The timer driver is a benchmark timer and is needed for the tmtests (timing tests). Sometimes you will see a shmsupp directory which is for shared memory multiprocessing systems. The network driver and real-time clock drivers are optional and not required by any RTEMS tests.

# 6 Debugging Hints

The questions in this category are hints that can ease debugging.

## 6.1 Executable Size

### 6.1.1 Why is my executable so big?

There are two primary causes for this. The most common is that you are doing an `ls -l` and looking at the actual file size – not the size of the code in the target image. This file could be in an object format such as ELF or COFF and contain debug information. If this is the case, it could be an order of magnitude larger than the required code space. Use the strip command in your cross toolset to remove debugging information.

The following example was done using the i386-rtems cross toolset and the pc386 BSP. Notice that with symbolic information included the file `hello.exe` is almost a megabyte and would barely fit on a boot floppy. But there is actually only about 93K of code and initialized data. The other 800K is symbolic information which is not required to execute the application.

```
$ ls -l hello.exe
-rwxrwxr-x    1 joel     users      930515 May  2 09:50 hello.exe
$ i386-rtems-size hello.exe
   text    data     bss     dec     hex filename
  88605    3591   11980  104176   196f0 hello.exe
$ i386-rtems-strip hello.exe
$ ls -l hello.exe
-rwxrwxr-x    1 joel     users      106732 May  2 10:02 hello.exe
$ i386-rtems-size hello.exe
   text    data     bss     dec     hex filename
  88605    3591   11980  104176   196f0 hello.exe
```

Another alternative is that the executable file is in an ASCII format such as Motorola Srecords. In this case, there is no debug information in the file but each byte in the target image requires two bytes to represent. On top of that, there is some overhead required to specify the addresses where the image is to be placed in target memory as well as checksum information. In this case, it is not uncommon to see executable files that are between two and three times larger than the actual space required in target memory.

Remember, the debugging information is required to do symbolic debugging with gdb. Normally gdb obtains its symbolic information from the same file that it gets the executable image from. However, gdb does not require that the executable image and symbolic information be obtained from the same file. So you might want to create a `hello_with_symbols.exe`, copy that file to `hello_without_symbols.exe`, and strip `hello_without_symbols.exe`. Then gdb would have to be told to read symbol information from `hello_with_symbols.exe`. The gdb command line option `-symbols` or command `symbol-file` may be used to specify the file read for symbolic information.

## 6.2 Malloc

### 6.2.1 Is malloc reentrant?

Yes. The RTEMS Malloc implementation is reentrant. It is implemented as calls to the Region Manager in the Classic API.

### 6.2.2 When is malloc initialized?

During BSP initialization, the `bsp_libc_init` routine is called. This routine initializes the heap as well as the RTEMS system call layer (open, read, write, etc.) and the RTEMS reentrancy support for the Cygnus newlib Standard C Library.

The `bsp_libc_init` routine is passed the size and starting address of the memory area to be used for the program heap as well as the amount of memory to ask `sbrk` for when the heap is exhausted. For most BSPs, all memory available is placed in the program heap thus it can not be extended dynamically by calls to `sbrk`.

## 6.3 How do I determine how much memory is left?

First there are two types of memory: RTEMS Workspace and Program Heap. The RTEMS Workspace is the memory used by RTEMS to allocate control structures for system objects like tasks and semaphores, task stacks, and some system data structures like the ready chains. The Program Heap is where "malloc'ed" memory comes from.

Both are essentially managed as heaps based on the Heap Manager in the RTEMS SuperCore. The RTEMS Workspace uses the Heap Manager directly while the Program Heap is actually based on an RTEMS Region from the Classic API. RTEMS Regions are in turn based on the Heap Manager in the SuperCore.

### 6.3.1 How much memory is left in the RTEMS Workspace?

An executive workspace overage can be fairly easily spotted with a debugger. Look at _Workspace_Area. If first == last, then there is only one free block of memory in the workspace (very likely if no task deletions). Then do this:

(gdb) p *(Heap_Block *)_Workspace_Area->first $3 = {back_flag = 1, front_flag = 68552, next = 0x1e260, previous = 0x1e25c}

In this case, I had 68552 bytes left in the workspace.

### 6.3.2 How much memory is left in the Heap?

The C heap is a region so this should work:

(gdb) p *((Region_Control *)_Region_Information->local_table[1])->Memory->first $9 = {back_flag = 1, front_flag = 8058280, next = 0x7ea5b4, previous = 0x7ea5b0}

In this case, the first block on the C Heap has 8,058,280 bytes left.

## 6.4 How do I convert an executable to IEEE-695?

This section is based on an email from Andrew Bythell <abythell@nortelnetworks.com> in July 1999.

Using Objcopy to convert m68k-coff to IEEE did not work. The new IEEE object could not be read by tools like the XRay BDM Debugger.

The exact nature of this problem is beyond me, but I did narrow it down to a problem with objcopy in binutils 2-9.1. To no surprise, others have discovered this problem as well, as it has been fixed in later releases.

I compiled a snapshot of the development sources from 07/26/99 and everything now works as it should. The development sources are at http://sourceware.cygnus.com/binutils (thanks Ian!)

Additional notes on converting an m68k-coff object for use with XRay (and others):

1. The m68k-coff object must be built with the -gstabs+ flag. The -g flag alone didn't work for me.
2. Run Objcopy with the –debugging flag to copy debugging information.

# 7 Free Software that Works with RTEMS

This section describes other free software packages that are known to work with RTEMS.

## 7.1 Development Tools

### 7.1.1 Basic Development Environment

The standard RTEMS development environment consists of the following GNU components:

- gcc
- binutils
- gdb

Although not from the Free Software Foundation, the Cygnus newlib C library integrates well with the GNU tools and is a standard part of the RTEMS development environment.

### 7.1.2 GNU Ada

For those interested in using the Ada95 programming language, the GNU Ada compiler (GNAT) is available and has excellent support for RTEMS.

### 7.1.3 DDD - Data Display Debugger

By far the easiest way to use DDD if you are on a Redhat or SuSE Linux system is to retrieve the RPM package for your OS version. In general, it is easier to install a static binary since doing so avoids all problems with dynamic library versions.

Some versions of DDD have had trouble with Lesstif. If you are using Lesstif, you will need version 0.88 or newer. It is also available as an RPM at the popular sites. Another Motif clone is Motive and versions 1.2 and newer known to work with DDD on popular distributions of Linux including RedHat and Slackware.

Installed as RPMs, DDD in conjunction with either Lesstif or Motive should work out-of-the-box.

User comments indicate that both Lesstif and DDD can be built from scratch without any problems. Instructions on installing DDD are at http://www.cs.tu-bs.de/softech/ddd/. They indicate that

> LessTif should be used in (default) Motif 1.2 compatibility mode.
>
> The Motif 2.0 compatibility mode of LessTif is still incomplete.

So configure lesstif with –enable-default-12.

The configure script is broken (see www.lesstif.org –> known problems) for 0.88.1. I didn't fix the script as they show, so I just have links in /usr/local/lib (also shown).

Watch out: Lesstif installs its libraries in /usr/local/Lesstif. You will need to update /etc/ld.so.conf and regenerate the cache of shared library paths to point to the Motif 1.2 library.

The following notes are from an RTEMS user who uses DDD in conjunction with Lesstif. Configure DDD "–with-motif-libraries=/usr/local/lib –with-motif-includes=/usr/local/include" DDD needs gnuplot 3.7. ftp://ftp.dartmouth.edu/pub/gnuplot/gnuplot-3.7.tar.gz. Build and install from scratch.

DDD can be started from a script that specifies the cross debugger. This simplifies the invocation. The following example shows what a script doing this looks like.

```
#!/bin/bash
ddd --debugger m68k-elf-gdb $1
```

Under many flavors of UNIX, you will likely have to relax permissions.

On Linux, to get gdb to use the serial ports while running as a normal user, edit /etc/security/console.perms, and create a <serial> class (call it whatever you want).

```
<serial>=/dev/ttyS* /dev/cua*
```

Now enable the change of ownership of these devices when users log in from the console:

```
<console> 0600 <serial> 0600 root
```

Users report using minicom to communicate with the target to initiate a TFTP download. They then suspend minicom, launch DDD, and begin debugging.

The procedure should be the same on other platforms, modulo the choice of terminal emulator program and the scheme used to access the serial ports. From problem reports on the cygwin mailing list, it appears that GDB has some problems communicating over serial lines on that platform.

NOTE: GDB does not like getting lots of input from the program under test over the serial line. Actually, it does not care, but it looses characters. It would appear that flow control is not re-enabled when it resumes program execution. At times, it looked like the test were failing, but everything was OK. We modified the MVME167 serial driver to send test output to another serial port. Using two serial ports is usually the easiest way to get test output while retaining a reliable debug connection regardless of the debugger/target combination.

NOTE: Enabling gdb's remote cache might prevent this (Observed with SH1 boards, but may also be valid for targets):

```
gdb > set remotecache
```

Information provided by Charles-Antoine Gauthier (charles.gauthier@iit.nrc.ca) Jiri Gaisler (jgais@ws.estec.esa.nl) and Ralf Corsépius (corsepiu@faw.uni-ulm.de)

## 7.2 omniORB

omniORB is a GPL'ed CORBA which has been ported to RTEMS. It is available from (http://www.uk.research.att.com/omniORB/omniORB.html) .

For information on the RTEMS port of omniORB to RTEMS, see the following URL (http://www.connecttel.com/corba/rtems_omni.html).

C++ exceptions must work properly on your target for omniORB to work.

The port of omniORB to RTEMS was done by Rosimildo DaSilva <rdasilva@connecttel.com>.

## 7.3 TCL

Tool Command Language.

ditto

## 7.4 ncurses

Free version of curses.

ditto

## 7.5 zlib

Free compression/decompression library.

ditto

# 8 Hardware to Ease Debugging

The items in this category provide information on various hardware debugging assistants that are available.

## 8.1 MC683xx BDM Support for GDB

Eric Norum (eric@skatter.usask.ca) has a driver for a parallel port interface to a BDM module. This driver has a long history and is based on a driver by Gunter Magin (magin@skil.camelot.de) which in turn was based on BD32 for DOS by Scott Howard. Eric Norum and Chris Johns (ccj@acm.org) have put together a package containing everything you need to use this BDM driver including software, PCB layouts, and machining drawings. From the README:

"This package contains everything you need to be able to run GDB on Linux and control a Motorola CPU32+ (68360) or Coldfire (5206, 5206e, or 5207) target through a standard PC parallel port."

Information on this BDM driver is available at the following URL:

`http://www.calm.hw.ac.uk/davidf/coldfire/gdb-bdm-linux.htm`

The package is officially hosted at Eric Norum's ftp site:

`ftp://skatter.usask.ca/pub/eric/BDM-Linux-gdb/`

Peter Shoebridge (peter@zeecube.com) has ported the Linux parallel port BDM driver from Eric Norum to Windows NT. It is available at http://www.zeecube.com/bdm.

The efi332 project has a home-built BDM module and gdb backend for Linux. See http://efi332.eng.ohio-state.edu/efi332/hardware.html) for details. The device driver and gdb backend are based on those by Gunter Magin (magin@skil.camelot.de) available from ftp.lpr.e-technik.tu-muenchen.de.

Pavel Pisa (pisa@cmp.felk.cvut.cz) has one available at http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.html.

Huntsville Microsystems (HMI) has GDB support for their BDM module available upon request. It is also available from their ftp site: ftp://ftp.hmi.com/pub/gdb

The Macraigor OCD BDM module has a driver for Linux written by Gunter Magin (magin@skil.camelot.de). No URLs yet.

Finally, there is a overview of BDM at the following URL: http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.html.

Information in this section from:

- Brendan Simon <brendan@dgs.monash.edu.au>
- W Gerald Hicks <wghicks@bellsouth.net>
- Chris Johns <ccj@acm.org>
- Eric Norum <eric@skatter.usask.ca>

- Gunter Magin <magin@skil.camelot.de>

## 8.2 MPC8xx BDM Support for GDB

Christian Haan <chn@intego.de> has written a driver for FreeBSD based for"a slightly changed ICD BDM module (because of changes in the BDM interface on the PowerPC)" that "probably will work with the PD module too." His work is based on the M68K BDM work by Gunter Magin (Gunter.Magin@skil.camelot.de) and the PPC BDM for Linux work by Sergey Drazhnikov (swd@agua.comptek.ru). This is not yet publicly available.

Sergey Drazhnikov (swd@agua.comptek.ru) has written a PPC BDM driver for Linux. Information is available at http://cyclone.parad.ru/ppcbdm.

Huntsville Microsystems (HMI) has GDB support for their BDM module available upon request. It is also available from their ftp site: ftp://ftp.hmi.com/pub/gdb

GDB includes support for a set of primitives to support the Macraigor Wiggler (OCD BDM). Unfortunately, this requires the use of a proprietary interface and is supported only on Windows. This forces one to use CYGWIN. Reports are that this results in a slow interface. Scott Howard (http://www.objsw.com) has announced that support for the gdb+wiggler combination under DJGPP which should run significantly faster.

- Leon Pollak <leonp@plris.com>
- Christian Haan <chn@intego.de>

# 9 RTEMS Projects

The questions in this category are regarding things that people are working on or that the RTEMS community would like to see work on.

There are multiple ways to support the RTEMS Project. Financial support is always welcomed. This can be via sponsorship of a specific project such as one of the ones listed here or via volunteering in some capacity.

## 9.1 Other Filesystems

This is a list of the filesystems that would be nice to have support for in RTEMS. For each filsystem, status and contact information is provided for those who have expressed interest in helping out or are actively working on it.

- TFTP client - read only, no write capability
    - No Contact
- DOS Filesystem - ???
    - Peter Shoebridge <peter@zeecube.com>
    - Victor V. Vengerov <vvv@tepkom.ru>
- CD-ROM Filesystem - ???
    - Peter Shoebridge <peter@zeecube.com>
- Flash Filesystem(s) - ???
    - Rod Barman <rodb@ieee.org>
    - Victor V. Vengerov <vvv@tepkom.ru>
- Remote Host Filesystem - ???
    - Wayne Bullaughey <wayne@wmi.com>
- NFS client - ???
    - No Contact

## 9.2 Java

### 9.2.1 Kaffe

This porting effort is underway and active.

- Jiri Gaisler <jgais@ws.estec.esa.nl>
- Oscar Martinez de la Torre <omt@wm.estec.esa.nl>

NOTE: An older version of Kaffe was ported to a pre-4.0 version of RTEMS. The network support in RTEMS was immature at that port and Kaffe had portability problems. Together these resulted in an unclean port which was never merged.

### 9.2.2 GNU Java Compiler (gjc)

This porting effort is underway and active.

- Charles-Antoine Gauthier <charles.gauthier@iit.nrc.ca>

## 9.3 CORBA

### 9.3.1 TAO

This porting effort is pending testing. Erik ported the code but then discovered that his target board did not have enough memory to run any TAO tests.

- Erik Ivanenko <erik.ivanenko@utoronto.ca>

## 9.4 APIs

### 9.4.1 POSIX 1003.1b

Support for POSIX 1003.1b is mature but there are a few remaining items including a handful of services, performance tests, and documentation. Please refer to the Status chapter of the *POSIX API User's Guide* for more details.

### 9.4.2 ITRON 3.0

Support for ITRON 3.0 is in its beginning stages. There are numerous managers left to implement, both functional and performance tests to write, and much documentation remaining. Please refer to the Status chapter of the *ITRON 3.0 API User's Guide* for specific details.

# 10 Date/Time Issues in Systems Using RTEMS

This section provides technical information regarding date/time representation issues and RTEMS. The Y2K problem has lead numerous people to ask these questions. The answer to these questions are actually more complicated than most people asking the question expect. RTEMS supports multiple standards and each of these standards has its own epoch and time representation. These standards include both programming API and programming language standards.

In addition to the issues inside RTEMS itself, there is the complicating factor that the Board Support Package or application itself may interface with hardware or software that has its own set of date/time representation issues.

In conclusion, viewing date/time representation as "the Y2K problem" is very short-sighted. Date/time representation should be viewed as a systems level issue for the system you are building. Each software and hardware component in the system as well as the systems being connected to is a factor in the equation.

## 10.1 Hardware Issues

Numerous Real-Time Clock (RTC) controllers provide only a two-digit Binary Coded Decimal (BCD) representation for the current year. Without software correction, these chips are a classic example of the Y2K problem. When the RTC rolls the year register over from 99 to 00, the device has no idea whether the year is 1900 or 2000. It is the responsibility of the device driver to recognize this condition and correct for it. The most common technique used is to assume that all years prior to either the existence of the board or RTEMS are past 2000. The starting year (epoch) for RTEMS is 1988. Thus,

- Chip year values 88-99 are interpreted as years 1988-2002.
- Chip year values 00-87 are interpreted as years 2000-2087.

Using this technique, a RTC using a two-digit BCD representation of the current year will overflow on January 1, 2088.

## 10.2 RTEMS Specific Issues

Internally, RTEMS uses an unsigned thirty-two bit integer to represent the number of seconds since midnight January 1, 1988. This counter will overflow on February 5, 2124.

The time/date services in the Classic API will overflow when the RTEMS internal date/time representation overflows.

The POSIX API uses the type *time_t* to represent the number of seconds since January 1, 1970. Many traditional UNIX systems as well as RTEMS define *time_t* as a signed thirty-two bit integer. This representation overflows on January 18, 2038. The solution usually proposed is to define *time_t* as a sixty-four bit integer. This solution is appropriate for for UNIX workstations as many of them already support sixty-four bit integers natively.

At this time, this imposes a burden on embedded systems which are still primarily using processors with native integers of thirty-two bits or less.

## 10.3 Language Specific Issues

The Ada95 Language Reference Manual requires that the *Ada.Calendar* package support years through the year 2099. However, just as the hardware is layered on top of hardware and may inherit its limits, the Ada tasking and run-time support is layered on top of an operating system. Thus, if the operating system or underlying hardware fail to correctly report dates after 2099, then it is possible for the *Ada.Calendar* package to fail prior to 2099.

## 10.4 Date/Time Conclusion

Each embedded system could be impacted by a variety of date/time representation issues. Even whether a particular date/time representation issue impacts a system is questionable. A system using only the RTEMS Classic API is not impacted by the date/time representation issues in POSIX. A system not using date/time at all is not impacted by any of these issues. Also the planned end of life for a system may make these issues moot.

The following is a timeline of the date/time representation issues presented in this section:

- 2000 - Two BCD Digit Real-Time Clock Rollover
- 2038 - POSIX *time_t* Rollover
- 2088 - Correction for Two BCD Digit Real-Time Clock Rollover
- 2099 - Ada95 *Ada.Calendar* Rollover
- 2124 - RTEMS Internal Seconds Counter Rollover