Getting Started with RTEMS

Edition 4.10.99.0, for 4.10.99.0

17 July 2015

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TEXinfo 2013-02-01.11

Table of Contents

1 Introduction

The purpose of this document is to guide you through the process of installing a GNU cross development environment to use with RTEMS.

If you are already familiar with the concepts behind a cross compiler and have a background in Unix, these instructions should provide the bare essentials for performing a setup of the following items:

- GNU Cross Compilation Tools for RTEMS on your build-host system
- RTEMS OS for the target
- GNU Debugger (GDB)

The remainder of this chapter provides background information on real-time embedded systems and cross development and an overview of other resources of interest on the Internet. If you are not familiar with real-time embedded systems or the other areas, please read those sections. These sections will help familiarize you with the types of systems RTEMS is designed to be used in and the cross development process used when developing RTEMS applications.

1.1 Real-Time Embedded Systems

Real-time embedded systems are found in practically every facet of our everyday lives. Today's systems range from the common telephone, automobile control systems, and kitchen appliances to complex air traffic control systems, military weapon systems, and production line control including robotics and automation. However, in the current climate of rapidly changing technology, it is difficult to reach a consensus on the definition of a real-time embedded system. Hardware costs are continuing to rapidly decline while at the same time the hardware is increasing in power and functionality. As a result, embedded systems that were not considered viable two years ago are suddenly a cost effective solution. In this domain, it is not uncommon for a single hardware configuration to employ a variety of architectures and technologies. Therefore, we shall define an embedded system as any computer system that is built into a larger system consisting of multiple technologies such as digital and analog electronics, mechanical devices, and sensors.

Even as hardware platforms become more powerful, most embedded systems are critically dependent on the real-time software embedded in the systems themselves. Regardless of how efficiently the hardware operates, the performance of the embedded real-time software determines the success of the system. As the complexity of the embedded hardware platform grows, so does the size and complexity of the embedded software. Software systems must routinely perform activities which were only dreamed of a short time ago. These large, complex, real-time embedded applications now commonly contain one million lines of code or more.

Real-time embedded systems have a complex set of characteristics that distinguish them from other software applications. Real-time embedded systems are driven by and must respond to real world events while adhering to rigorous requirements imposed by the environment with which they interact. The correctness of the system depends not only on the results of computations, but also on the time at which the results are produced. The most

important and complex characteristic of real-time application systems is that they must receive and respond to a set of external stimuli within rigid and critical time constraints.

A single real-time application can be composed of both soft and hard real-time components. A typical example of a hard real-time system is a nuclear reactor control system that must not only detect failures, but must also respond quickly enough to prevent a meltdown. This application also has soft real-time requirements because it may involve a man-machine interface. Providing an interactive input to the control system is not as critical as setting off an alarm to indicate a failure condition. However, the interactive system component must respond within an acceptable time limit to allow the operator to interact efficiently with the control system.

1.2 Cross Development

Today almost all real-time embedded software systems are developed in a **cross development** environment using cross development tools. In the cross development environment, software development activities are typically performed on one computer system, the **build-host** system, while the result of the development effort (produced by the cross tools) is a software system that executes on the **target** platform. The requirements for the target platform are usually incompatible and quite often in direct conflict with the requirements for the build-host. Moreover, the target hardware is often custom designed for a particular project. This means that the cross development toolset must allow the developer to customize the tools to address target specific run-time issues. The toolset must have provisions for board dependent initialization code, device drivers, and error handling code.

The build-host computer is optimized to support the code development cycle with support for code editors, compilers, and linkers requiring large disk drives, user development windows, and multiple developer connections. Thus the build-host computer is typically a traditional UNIX workstation such as those available from SUN or Silicon Graphics, or a PC running either a version of MS-Windows or UNIX. The build-host system may also be required to execute office productivity applications to allow the software developer to write documentation, make presentations, or track the project's progress using a project management tool. This necessitates that the build-host computer be general purpose with resources such as a thirty-two or sixty-four bit processor, large amounts of RAM, a monitor, mouse, keyboard, hard and floppy disk drives, CD-ROM drive, and a graphics card. It is likely that the system will be multimedia capable and have some networking capability.

Conversely, the target platform generally has limited traditional computer resources. The hardware is designed for the particular functionality and requirements of the embedded system and optimized to perform those tasks effectively. Instead of hard drives and keyboards, it is composed of sensors, relays, and stepper motors. The per-unit cost of the target platform is typically a critical concern. No hardware component is included without being cost justified. As a result, the processor of the target system is often from a different processor family than that of the build-host system and usually has lower performance. In addition to the processor families designed only for use in embedded systems, there are versions of nearly every general-purpose processor specifically tailored for real-time embedded systems. For example, many of the processors targeting the embedded market do not include hardware floating point units, but do include peripherals such as timers, serial controllers, or network interfaces.

1.3 Resources on the Internet

This section describes various resources on the Internet which are of use to RTEMS users.

1.3.1 Online Tool Documentation

Each of the tools in the GNU development suite comes with documentation. It is in the reader's and tool maintainers' interest that one read the documentation before posting a problem to a mailing list or news group. The RTEMS Project provides formatted documentation for the primary tools in the cross development toolset including BINUTILS, GCC, NEWLIB, and GDB with the pre-built versions of those tools.

Much of the documentation is available at other sites on the Internet, for example the GNU manuals are hosted by the Free Software Foundation at http://www.gnu.org/manual/manual.html.

1.3.2 RTEMS Mailing Lists

users@rtems.org

The users mailing list is for any and all questions about RTEMS, especially those focusing on how to use RTEMS. If you would like to browse the thousands of messages in the fifteen year archive of the mailing list or subscribe to it, please visit https://lists.rtems.org/mailman/listinfo/users for more information,

devel@rtems.org

The devel mailing list is the place to track ongoing RTEMS development and to discuss changes to RTEMS. This list is also where patches are submitted.

2 Requirements

This chapter describes the build-host system requirements and initial steps in installing the GNU Cross Compiler Tools and RTEMS on a build-host.

2.1 Disk Space

A fairly large amount of disk space is required to perform the build of the GNU C/C++ Cross Compiler Tools for RTEMS. The following table may help in assessing the amount of disk space required for your installation:

Component	Disk Space Required
archive directory	120 Mbytes
tools src unarchived	1400 Mbytes
each individual build directory	up to 2500 Mbytes
each installation directory	900 Mbytes

It is important to understand that the above requirements only address the GNU C/C++ Cross Compiler Tools themselves. Adding additional languages such as Ada or Go can increase the size of the build and installation directories. Also, the unarchived source and build directories can be removed after the tools are installed.

After the tools themselves are installed, RTEMS must be built and installed for each Board Support Package that you wish to use. Thus the precise amount of disk space required for each installation directory depends highly on the number of RTEMS BSPs which are to be installed. If a single BSP is installed, then the additional size of each install directory will tend to be in the 40-60 Mbyte range.

There are a number of factors which must be taken into account in order to estimate the amount of disk space required to build RTEMS itself. Attempting to build multiple BSPs in a single step increases the disk space requirements. On some target architectures, this can lead to disk usage during the build of over one gigabyte.

Similarly enabling optional features increases the build and install space requirements. In particular, enabling and building the RTEMS tests results in a significant increase in build space requirements but since the tests are not installed has, enabling them has no impact on installation requirements.

2.2 General Host Software Requirements

The instructions in this manual should work on any computer running a POSIX environment including GNU/Linux and Cygwin. Mingw users may encounter additional issues due to the limited POSIX compatibility. Some native GNU tools are used by this procedure including:

- GCC
- GNU make
- GNU makeinfo

In addition, some native utilities may be deficient for building the GNU tools. On hosts which have m4 but it is not GNU m4, it is not uncommon to have to install GNU m4. Similarly, some shells are not capable of fully supporting the RTEMS configure scripts.

2.2.1 GCC

Although RTEMS itself is intended to execute on an embedded target, there is source code for some native programs included with the RTEMS distribution. Some of these programs are used to assist in the building of RTEMS itself, while others are BSP specific tools. Regardless, no attempt has been made to compile these programs with a non-GNU compiler.

2.2.2 GNU Make

Both NEWLIB and RTEMS use GNU make specific features and can only be built using GNU make. Many systems include a make utility that is not GNU make. The safest way to meet this requirement is to ensure that when you invoke the command `make`, it is GNU make. This can be verified by attempting to print the GNU make version information:

```
make --version
```

If you have GNU make and another make on your system, it is common to put the directory containing GNU make before the directory containing other implementations of make.

2.2.3 GNU makeinfo Version Requirements

In order to build gcc 2.9.x or newer versions, the GNU `makeinfo` program installed on your system must be at least version 1.68. The appropriate version of `makeinfo` is distributed with `gcc`.

The following demonstrates how to determine the version of `makeinfo` on your machine:

```
makeinfo --version
```

2.3 Host Specific Notes

2.3.1 Solaris 2.x

The following problems have been reported by Solaris 2.x users:

- The build scripts are written in "shell". The program `/bin/sh` on Solaris 2.x is not robust enough to execute these scripts. If you are on a Solaris 2.x host, then use the `/bin/ksh` or `/bin/bash` shell instead.
- The native `patch` program is broken. Install the GNU version.
- The native `m4` program is deficient. Install the GNU version.

2.3.2 Distribution Independent Potential GNU/Linux Issues

The following problems have been reported by users of various GNU/Linux distributions:

- Certain versions of GNU fileutils include a version of `install` which does not work properly. Please perform the following test to see if you need to upgrade:

  ```
  install -c -d /tmp/foo/bar
  ```

If this does not create the specified directories your install program will not install RTEMS properly. You will need to upgrade to at least GNU fileutils version 3.16 to resolve this problem.

3 Building the GNU Cross Compiler Toolset with RSB

The RTEMS Projects recommends using the RTEMS Source Builder (RSB) for building the toolset from source. RSB has evolved over time from various instructions and scripts for building the toolset, and it removes much of the frustration associated with building the toolset from source. Although prebuilt binaries are much easier to install, they are harder for the RTEMS Project to support.

Documentation for RSB is available from https://docs.rtems.org/rsb/.

4 Building RTEMS

NOTE: If you built your toolset with RSB, by default the RSB also builds RTEMS while building the compiler toolset. You may already have a built and installed RTEMS in this case, and if not you should check the RSB documentation at https://docs.rtems.org/rsb/.

4.1 Obtain the RTEMS Source Code

This section provides pointers to the RTEMS source code and example programs. These files should be placed in your `archive` directory. The set of tarballs which comprise an RTEMS release is placed in a directory whose name is the release on the ftp site. The RTEMS ftp site is accessible via both the ftp and http protocols at the following URLs:

- http://ftp.rtems.org/pub/rtems
- ftp://ftp.rtems.org/pub/rtems

Associated with each RTEMS Release is a set of example programs. Prior to the 4.10 Release Series, these examples were in a "Class Examples" and an "Examples" collection. Beginning with the 4.10 Release Series, these examples collections were merged and other examples added. This new collection is called "Examples V2". It is contained in the file `examples-v2-<VERSION>.tar.bz2>` within the RTEMS release directory.

4.2 Unarchive the RTEMS Source

Use the following command sequence to unpack the RTEMS source into the tools directory:

```
cd tools
tar xjf ../archive/rtems-4.12.<VERSION>.tar.bz2
```

This creates the directory rtems-4.12.<VERSION>

4.3 Obtaining the RTEMS Source from Git

Instead of downloading release tarballs you may choose to check out the current RTEMS source from the project's source code repository. For details on accessing the RTEMS source repository consult:

https://devel.rtems.org/wiki/Developer/Git.

4.4 Add <INSTALL_POINT>/bin to Executable PATH

In order to compile RTEMS, you must have the cross compilation toolset in your search path. It is important to have the RTEMS toolset first in your path to ensure that you are using the intended version of all tools. The following command prepends the directory where the tools were installed in a previous step. If you are using binaries installed to `/opt/rtems-4.12`, then the <INSTALL_POINT> will be `/opt/rtems-4.12`

```
export PATH=<INSTALL_POINT>/bin:${PATH}
```

NOTE: The above command is in Bourne shell (`sh`) syntax and should work with the Korn (`ksh`) and GNU Bourne Again Shell (`bash`). It will not work with the C Shell (`csh`) or derivatives of the C Shell.

4.5 Verifying the Operation of the Cross Toolset

In order to ensure that the cross-compiler is invoking the correct subprograms (like `as` and `ld`), one can test assemble a small program. When in verbose mode, `gcc` prints out information showing where it found the subprograms it invokes. In a temporary working directory, place the following function in a file named `f.c`:

```
int f( int x )
{
  return x + 1;
}
```

Then assemble the file using a command similar to the following:

```
m68k-rtems4.12-gcc -v -S f.c
```

Where `m68k` should be changed to match the target architecture of your cross compiler. The result of this command will be a sequence of output showing where the cross-compiler searched for and found its subcomponents. Verify that these paths correspond to your <INSTALL_POINT>.

Look at the created file `f.s` and verify that it is in fact for your target processor.

Then try to compile the file `f.c` directly to object code using a command like the following:

```
m68k-rtemsRTEMSAPI-gcc -v -c f.c
```

If this produces messages that indicate the assembly code is not valid, then it is likely that you have fallen victim to one of the most common installation errors and the cross-compiler is not able to find the cross assembler and defaults to using the native `as`. This can result in very confusing error messages.

4.6 Building RTEMS for a Specific Target and BSP

This section describes how to configure and build RTEMS so that it is specifically tailored for your BSP (Board Support Package) and the CPU model it uses. There is currently only one supported method to compile and install RTEMS:

- direct invocation of `configure` and `make`

Direct invocation of `configure` and `make` provides more control and easier recovery from problems when building.

This section describes how to build RTEMS.

4.6.1 Using the RTEMS configure Script Directly

Make a build directory under tools and build the RTEMS product in this directory. The `../rtems-4.12.<VERSION>/configure` command has numerous command line arguments. These arguments are discussed in detail in documentation that comes with the RTEMS distribution. A full list of these arguments can be obtained by running `../rtems-4.12.<VERSION>/configure --help` If you followed the procedure described in the section Section 4.2 [Unarchive the RTEMS Source], page 11 or Section 4.3 [Obtaining the RTEMS

Source from Git], page 11, these configuration options can be found in the file rtems-4.12.<VERSION>/README.configure.

NOTE: The GNAT/RTEMS run-time implementation is based on the POSIX API and the GNAT/RTEMS run-time cannot be compiled with networking disabled. Your application does not have to use networking but it must be enabled. Thus the RTEMS configuration for a GNAT/RTEMS environment MUST include the `--enable-posix --enable-networking` flag.

NOTE: Building RTEMS requires that a few support programs be compiled natively. This means there must be a native toolchain installed on your development host. You will need to have a native compiler such as *gcc* or *cc* in your execution path. If you cannot compile, link and execute a native hello world program, then you will be unable to build RTEMS.

The following shows the command sequence required to configure, compile, and install RTEMS with the POSIX API, FreeBSD TCP/IP, and C++ support disabled. RTEMS will be built to target the `BOARD_SUPPORT_PACKAGE` board.

```
mkdir build-rtems
cd build-rtems
../rtems-4.12.VERSION/configure \
    --target=<TARGET_CONFIGURATION> \
    --disable-networking \
    --enable-rtemsbsp=<BSP>\
    --prefix=<INSTALL_POINT>
make all
make install
```

<TARGET> is of the form <CPU>-rtems4.12 and the list of currently supported <TARGET> configuration's and <BSP>'s can be found in `tools/RTEMS-4.12.<VERSION>/README.configure`.

<INSTALL_POINT> is typically the installation point for the tools and defaults to `/opt/rtems-4.12`.

BSP is a supported BSP for the selected CPU family. The list of supported BSPs may be found in the file `tools/rtems-4.12.<VERSION>/README.configure` in the RTEMS source tree. If the BSP parameter is not specified, then all supported BSPs for the selected CPU family will be built.

NOTE: The POSIX API and networking must be enabled to use GNAT/RTEMS.

NOTE: The `make` utility used should be GNU make.

5 Building the Sample Applications

The RTEMS distribution includes a number of sample C, C++, Ada, and networking applications. This chapter will provide an overview of those sample applications.

5.1 Set the Environment Variable RTEMS_MAKEFILE_PATH

The sample application sets use the RTEMS Application Makefiles. This requires that the environment variable `RTEMS_MAKEFILE_PATH` point to the appropriate directory containing the installed RTEMS image built to target your particular CPU and board support package combination.

```
export RTEMS_MAKEFILE_PATH=<INSTALLATION_POINT>/<CPU>-rtems/<BOARD_SUPPORT_PACKAGE>
```

Where <INSTALLATION_POINT> and <BOARD_SUPPORT_PACKAGE> are those used when configuring and installing RTEMS.

NOTE: In release 4.0, BSPs were installed at `<INSTALLATION_POINT>/rtems/<BOARD_SUPPORT_PACKAGE>`. This was changed to be more in compliance with GNU standards.

NOTE: GNU make is the preferred `make` utility. Other `make` implementations may work but all testing is done with GNU make.

If no errors are detected during the sample application build, it is reasonable to assume that the build of the GNU Cross Compiler Tools for RTEMS and RTEMS itself for the selected host and target combination was done properly.

5.2 Executing the Sample Applications

How each sample application executable is downloaded to your target board and executed is very dependent on the board you are using. The following is a list of commonly used BSPs classified by their RTEMS CPU family and pointers to instructions on how to use them. [NOTE: All file names should be prepended with `rtems-4.12.<VERSION>/c/src/lib/libbsp`.]

arm/edp7312 The arm/edp7312 BSP is for the ARM7-based Cogent EDP7312 board.

c4x/c4xsim The c4x/c4xsim BSP is designed to execute on any member of the Texas Instruments C3x/C4x DSP family using only on-CPU peripherals for the console and timers.

i386/pc386 See `i386/pc386/HOWTO`

i386/pc486 The i386/pc386 BSP specially compiled for an i486-class CPU.

i386/pc586 The i386/pc386 BSP specially compiled for a Pentium-class CPU.

i386/pc686 The i386/pc386 BSP specially compiled for a Pentium II.

i386/pck6 The i386/pc386 BSP specially compiled for an AMD K6.

m68k/gen68360 This BSP is for a MC68360 CPU. See `m68k/gen68360/README` for details.

m68k/mvme162 See `m68k/mvme162/README`.

m68k/mvme167 See `m68k/mvme167/README`.

mips/jmr3904 This is a BSP for the Toshiba TX3904 evaluation board simulator included with `mipstx39-rtems-gdb`. The BSP is located in `mips/jmr3904`. The TX3904 is a MIPS R3000 class CPU with serial ports and timers integrated with the processor. This BSP can be used with either real hardware or with the simulator included with `mipstx39-rtems-gdb`. An application can be run on the simulator by executing the following commands upon entering `mipstx39-rtems-gdb`:

```
target sim --board=jmr3904
load
run
```

powerpc/mcp750 See `powerpc/motorola_shared/README`.

powerpc/mvme230x See `powerpc/motorola_shared/README.MVME2300`.

powerpc/psim This is a BSP for the PowerPC simulator included with `powerpc-rtems-gdb`. The simulator is complicated to initialize by hand. The user is referred to the script `powerpc/psim/tools/psim`.

sparc/erc32 The ERC32 is a radiation hardened SPARC V7. This BSP can be used with either real ERC32 hardware or with the simulator included with `sparc-rtems-gdb` (for this, you should configure RTEMS to use `sis` BSP). An application can be run on the simulator by executing the following commands upon entering `sparc-rtems-gdb`:

```
target sim
load
run
```

In case that you don't need a debugger, an application can be run by `spart-rtems-run`.

RTEMS has many more BSPs and new BSPs for commercial boards and CPUs with on-CPU peripherals are generally welcomed.

5.3 C/C++ Sample Applications

The C/C++ sample application set includes a number of simple applications. Some demonstrate some basic functionality in RTEMS such as writing a file, closing it, and reading it back while others can serve as starting points for RTEMS applications or libraries. Start by unarchiving them so you can peruse them. Use a command similar to the following to unarchive the sample applications:

```
cd tools
tar xjf ../archive/examples-v2-4.12.<VERSION>.tgz
```

Each tests is found in a separate subdirectory and built using the same command sequence. The `hello/hello_world_c` sample will be used as an example.

Build the C Hello World Application

Use the following command to start the build of the sample hello world application:

```
cd hello_world_c
make
```

If the sample application has successfully been built, then the application executable is placed in the following directory:

```
hello_world_c/o-optimize/<filename>.ralf
```

The other sample applications are built using a similar procedure.

5.4 Ada Sample Applications

The Ada sample application set primarily includes a a simple Hello World Ada program which can be used as a starting point for GNAT/RTEMS applications. Use the following command to unarchive the Ada sample applications:

```
cd tools
tar xjf ../archive/ada-examples-4.12.<VERSION>.tgz
```

Create a BSP Specific Makefile

Currently, the procedure for building and linking an Ada application is a bit more difficult than a C or C++ application. This is certainly an opportunity for a volunteer project.

If your BSP requires special arguments when linking, you may have to augment the file `ada-examples-4.12.<VERSION>/Makefile.shared`. Most RTEMS BSPs do not require special linking arguments so this should not be frequently needed.

Use the <INSTALLATION_POINT> and <BOARD_SUPPORT_PACKAGE> specified when configuring and installing RTEMS.

5.5 Build the Sample Application

Use the following command to start the build of the sample application:

```
cd tools/ada-examples-4.12.<VERSION>/ada-examples/hello_world_ada
```

If no errors are detected during the sample application build, it is reasonable to assume that the build of the GNAT/RTEMS Cross Compiler Tools for RTEMS and RTEMS itself for the selected host and target combination was done properly.

5.6 Application Executable

If the sample application has successfully been build, then the application executable is placed in the following directory:

```
tools/ada-examples-4.12.<VERSION>/hello_world_ada/o-optimize/<filename>.exe
```

How this executable is downloaded to the target board is very dependent on the BOARD_SUPPORT_PACKAGE selected.

5.7 More Information on RTEMS Application Makefiles

These sample applications are examples of simple RTEMS applications that use the RTEMS Application Makefile system. This Makefile system simplifies building RTEMS applications by providing Makefile templates and capturing the configuration information used to build RTEMS specific to your BSP. Building an RTEMS application for different BSPs is as simple as switching the setting of `RTEMS_MAKEFILE_PATH`. This Makefile system is described in the file `make/README`.

It is very likely in the future that the RTEMS examples built using an installed RTEMS will be converted to autoconf.

6 Where To Go From Here

At this point, you should have successfully installed a GNU Cross Compilation Tools for RTEMS on your host system as well as the RTEMS OS for the target host. You should have successfully linked the "hello world" program. You may even have downloaded the executable to that target and run it. What do you do next?

The answer is that it depends. You may be interested in writing an application that uses one of the multiple APIs supported by RTEMS. You may need to investigate the network or filesystem support in RTEMS. The common thread is that you are largely finished with this manual and ready to move on to others.

Whether or not you decide to dive in now and write application code or read some documentation first, this chapter is for you. The first section provides a quick roadmap of some of the RTEMS documentation. The next section provides a brief overview of the RTEMS application structure.

6.1 Documentation Overview

When writing RTEMS applications, you should find the following manuals useful because they define the calling interface to many of the services provided by RTEMS:

- **RTEMS Applications C User's Guide** describes the Classic RTEMS API based on the RTEID specification.
- **RTEMS POSIX API User's Guide** describes the RTEMS POSIX API that is based on the POSIX 1003.1b API. If there is any place where this manual is thin or unclear, please refer to the OpenGroup Single UNIX Specification. RETEMS tracks that specification for future POSIX revisions.
- **RTEMS Network Supplement** provides information on the network services provided by RTEMS. RTEMS provides a BSD sockets programming interface so any network programming book should be helpful.

In addition, the following manuals from the GNU Cross Compilation Toolset include information on run-time services available.

- **C Support Library** describes the Standard C Library functionality provided by Newlib's libc.
- **C Math Library** describes the Standard C Math Library functionality provided by Newlib's libm.

Finally, the RTEMS FAQ, Wiki, and mailing list archives are available at http://www.rtems.org.

There is a wealth of documentation available for RTEMS and the GNU tools supporting it. If you run into something that is not clear or missing, bring it to our attention.

Also, some of the RTEMS documentation is still under construction. If you would like to contribute to this effort, please contact the RTEMS Team at users@rtems.org. If you are interested in sponsoring the development of a new feature, BSP, device driver, port of an existing library, etc., please contact sales@oarcorp.com.

6.2 Writing an Application

From an application author's perspective, the structure of an RTEMS application is very familiar. In POSIX language, RTEMS provides a single process, multi-threaded run-time environment. However there are two important things that are different from a standard UNIX hosted program.

First, the application developer must provide configuration information for RTEMS. This configuration information includes limits on the maximum number of various OS resources available and networking configuration among other things. See the **Configuring a System** in the **RTEMS Applications C User's Guide** for more details.

Second, RTEMS applications may or may not start at `main()`. Applications begin execution at one or more user configurable application initialization tasks or threads. It is possible to configure an application to start with a single thread that whose entry point is `main()`.

Each API supported by RTEMS (Internal, Classic, and POSIX) allows the user to configure a set of one or more tasks that are created and started automatically during RTEMS initialization. The RTEMS Automatic Configuration Generation (`confdefs.h`) scheme can be used to easily generate the configuration information for an application that starts with a single initialization task. By convention, unless overridden, the default name of the initialization task varies based up API.

- `Init` - single Classic API Initialization Task
- `POSIX_Init` - single POSIX API Initialization Thread

Regardless of the API used, when the initialization task executes, all non-networking device drivers are normally initialized, processor interrupts are enabled, and any C++ global constructors have been run. The initialization task then goes about its business of performing application specific initialization which will include initializing the networking subsystem if it is to be used. The application initialization may also involve creating tasks and other system resources such as semaphores or message queues and allocating memory. In the RTEMS examples and tests, the file `init.c` usually contains the initialization task. Although not required, in most of the examples, the initialization task completes by deleting itself.

As you begin to write RTEMS application code, you may be confused by the range of alternatives. Supporting multiple tasking APIs can make the choices confusing. Many application groups writing new code choose one of the APIs as their primary API and only use services from the others if nothing comparable is in their preferred one. However, the support for multiple APIs is a powerful feature when integrating code from multiple sources. You can write new code using POSIX services and still use services written in terms of the other APIs. Moreover, by adding support for yet another API, one could provide the infrastructure required to migrate from a legacy RTOS with a non-standard API to an API like POSIX.

Appendix A Using MS-Windows as a Development Host

This chapter discusses the installation of the GNU tool chain on a computer running the Microsoft Windows operating system.

This chapter was originally written by Geoffroy Montel <g_montel@yahoo.com> with input from David Fiddes <D.J@fiddes.surfaid.org>. It was based upon his successful but unnecessarily painful efforts with Cygwin beta versions. Cygwin and this chapter have been updated multiple times since those early days although their pioneering efforts and input is still greatly appreciated.

A.1 Microsoft Windows Version Requirements

RTEMS users report fewer problems when using Microsoft Windows XP or newer.

A.2 Cygwin

For RTEMS development, the recommended approach is to use Cygwin. Cygwin is available from http://www.cygwin.com . The primary issues reported by users of Cygwin is that it is slower on the same hardware than a native GNU/Linux installation and strange issues over carriage return/line feed inconsistencies between UNIX and Windows environments. However, there are a handful of other issues that may turn up when using Cygwin as an RTEMS development environment.

- There is no `cc` program by default. The GNU configure scripts used by RTEMS require this to be present to work properly. The solution is to link `gcc.exe` to `cc.exe` as follows:

  ```
  ln -s /bin/gcc.exe  /bin/cc.exe
  ```

- Make sure `/bin/sh.exe` is GNU Bash. Some Cygwin versions provide a light Bourne shell which is insufficient to build RTEMS. To see which shell is installed as `/bin/sh.exe`, execute the command `/bin/sh --version`. If it looks similar to the following, then it is GNU Bash and you are OK:

  ```
  GNU bash, version 2.04.5(12)-release (i686-pc-cygwin)
  Copyright 1999 Free Software Foundation, Inc.
  ```

  If you get an error or it claims to be any other shell, you need to copy it to a fake name and copy `/bin/bash.exe` to `/bin/sh.exe`:

  ```
  cd /bin
  mv sh.exe old_sh.exe
  cp bash.exe sh.exe
  ```

  The Bourne shell has to be present in `/bin` directory to run shell scripts properly.

- Make sure you unarchive and build in a binary mounted filesystem (e.g. mounted with the `-b` option). Otherwise, many confusing errors will result.
- A user has reported that they needed to set `CYGWIN=ntsec` for chmod to work correctly, but had to set `CYGWIN=nontsec` for compile to work properly (otherwise there were complaints about permissions on a temporary file).
- If you want to build the tools from source, you have the same options as UNIX users.

- You may have to uncompress archives during this process. You must **NOT** use `WinZip` or `PKZip`. Instead the un-archiving process uses the GNU `zip` and `tar` programs as shown below:

  ```
  tar -xzvf archive.tgz
  ```

  `tar` is provided with Cygwin.

A.3 MingGW

You might consider choosing MinGW since it provides better perfomance. There is a wiki entry on a MinGW RTEMS toolset installer available at http://www.rtems.org/wiki/index.php/MinGW_Tools_for_Windows. Also, there are prebuilt tools for different architectures available for download at http://www.rtems.org/ftp/pub/rtems/mingw32.

A.4 Text Editor

You absolutely have to use a text editor which can save files with Unix format. So do **NOT** use Notepad or Wordpad! Basically, any more or less advanced text editor is usually able to do that. There is a number of editors freely available that can be used.

- **Notepad++** has an interface familiar to Windows users and can be downloaded from http://notepad-plus-plus.org/.
- **VIM** (**Vi IMproved**) is available from http://www.vim.org/. This editor has the very handy ability to easily read and write files in either DOS or UNIX style.
- **GNU Emacs** is available for many platforms including MS-Windows. The official homepage is http://www.gnu.org/software/emacs/emacs.html. The GNU Emacs on Windows NT and Windows 95/98 FAQ is at http://www.gnu.org/software/emacs/windows/ntemacs.html.

If you do accidentally end up with files having MS-DOS style line termination, then you may have to convert them to Unix format for some Cygwin programs to operate on them properly. The program `dos2unix` can be used to put them back into Unix format as shown below:

```
$ dos2unix XYZ
Dos2Unix: Cleaning file XYZ ...
```

A.5 System Requirements

Although the finished cross-compiler is fairly easy on resources, building it can take a significant amount of processing power and disk space. Luckily, desktop computers have progressed very far since this guide was originally written so it is unlikely you will have any problems. Just do not use an old cast-off machine with < 1 GB RAM and a 1 Ghz CPU. Unless, of course, you enjoy waiting for things to complete.

The more disk space, the better. You need more if you are building the GNU tools and the amount of disk space for binaries is obviously directly dependent upon the number of CPUs you have cross toolsets installed for. In addition to the disk space requirements documented earlier for tool building, you will also have to have enough space to install the Cygwin environment.