Getting Started with RTEMS for C/C++ Users

Edition 4.0.0, for RTEMS 4.0.0

October 1998

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TEXinfo 1.1.1.1

Any inquiries concerning RTEMS, its related support components, or its documentation should be directed to either:

```
On-Line Applications Research Corporation
4910-L Corporate Drive
Huntsville, AL 35805
VOICE: (256) 722-9985
FAX:   (256) 722-0985
EMAIL: rtems@OARcorp.com
```

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 C/C++ Cross Compilation Tools for RTEMS on your host system
- RTEMS OS for the target host
- GDB Debugger

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, an d 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, th e 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 **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 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 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 host computer is typically a traditional UNIX workstation such as are available from SUN or Silicon Graphics, or a PC running either a version of MS-Windows or UNIX. The 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 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 driversΦΦs 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 host system and usually has lower performance. In addition to the processor families targeted only for use in embedded systems, there are versions of nearly every general-purpose process or 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 RTEMS Mailing List

rtems-list@OARcorp.com

This mailing list is dedicated to discussion of issues related to RTEMS. If you have questions about RTEMS, wish to make suggestions, or just want to pick up hints, this is a good list to subscribe to. Subscribe by sending a message with the one line "subscribe" to rtems-list-request@OARcorp.com.

1.3.2 CrossGCC Mailing List

crossgcc@cygnus.com

This mailing list is dedicated to the use of the GNU tools in cross development environments. Most of the discussions focus on embedded issues. Subscribe by sending a message with the one line "subscribe" to crossgcc-request@cygnus.com.

The crossgcc FAQ as well as a number of patches and utiliities of interest to cross development system users are available at ftp://ftp.cygnus.com/pub/embedded/crossgcc.

1.3.3 EGCS Mailing List

egcs@cygnus.com

This mailing list is dedicated to the EGCS Project which was formed to speed the development and integration of the various GNU languages. The RTEMS and Linux communities were among those initially targetted by the EGCS Project as being important for its success. Numerous RTEMS users have made contributions to this project. Subscribe by sending a message with the one line "subscribe" to egcs-request@cygnus.com.

2 Requirements

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	30 Mbytes
tools src unzipped	100 Mbytes
each individual build directory	300 Mbytes worst case
each installation directory	20-400 Mbytes

The disk space required for each installation directory depends primarily on the number of RTEMS BSPs which are to be installed. If a single BSP is installed, then the size of each install directory will tend to be in the 40-60 Mbyte range.

The instructions in this manual should work on any computer running a UNIX variant. 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.

2.1 GNU makeinfo Version Requirements

In order to build egcs 1.1b or newer, the GNU `makeinfo` program installed on your system must be at least version 1.68. The appropriate version of `makeinfo` is distributed with egcs 1.1b.

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

```
makeinfo --version
```

3 Building the GNU C/C++ Cross Compiler Toolset

This chapter describes the steps required to acquire the source code for a GNU cross compiler toolset, apply any required RTEMS specific patches, compile that toolset and install it.

3.1 Create the Archive and Build Directories

Start by making the `archive` directory to contain the downloaded source code and the `tools` directory to be used as a build directory. The command sequence to do this is shown below:

```
mkdir archive
mkdir tools
```

This will result in an initial directory structure similar to the one shown in the following figure:

```
/whatever/prefix/you/choose/
        archive/
        tools/
```

3.2 Get All the Pieces

This section lists the components of an RTEMS cross development system. Included are the locations of each component as well as any required RTEMS specific patches.

egcs 1.1b

```
FTP Site:    egcs.cygnus.com
Directory:   /pub/egcs/releases/egcs-1.1b
File:        egcs-1.1b.tar.gz
```

binutils 2.9.1

```
FTP Site:    ftp.gnu.org
Directory:   /pub/gnu
File:        binutils-2.9.1.tar.gz
```

newlib 1.8.0

```
FTP Site:    ftp.cygnus.com
Directory:   /pub/newlib
File:        newlib-1.8.0.tar.gz
```

RTEMS 4.0.0

```
FTP Site:    ftp.OARcorp.com
Directory:   /pub/rtems/4.0.0
File:        rtems-4.0.0.tgz
```

RTEMS Hello World

```
FTP Site:    ftp.OARcorp.com
Directory:   /pub/rtems/4.0.0
File:        hello_world_c.tgz
```

RTEMS Specific Tool Patches and Scripts

```
FTP Site:    ftp.OARcorp.com
Directory:   /pub/rtems/4.0.0/c_tools
File:        c_build_scripts-4.0.0.tgz
File:        binutils-2.9.1-rtems-diff-19981027.gz
File:        newlib-1.8.0-rtems-diff-19981027.gz
File:        egcs-1.1b-rtems-diff-19981027.gz
```

3.3 Unarchiving the Tools

While in the `tools` directory, unpack the compressed tar files using the following command sequence:

```
cd tools
tar xzf ../archive/egcs-1.1b.tar.gz
tar xzf ../archive/binutils-2.9.1.tar.gz
tar xzf ../archive/newlib-1.8.0.tar.gz
tar xzf ../archive/c_build_scripts-4.0.0.tgz
```

After the compressed tar files have been unpacked, the following directories will have been created under tools.

- binutils-2.9.1
- egcs-1.1b
- newlib-1.8.0

There will also be a set of scripts in the current directory which aid in building the tools and RTEMS. They are:

- bit
- bit_gdb
- bit_rtems
- common.sh
- user.cfg

When the `bit` script is executed later in this process, it will automatically create two other subdirectories:

- src

- build-${CPU}-tools

Similarly, the `bit_gdb` script will create the subdirectory `build-${CPU}-gdb` and the `bit_rtems` script will create the subdirectory `build-${CPU}-rtems`.

The tree should look something like the following figure:

```
/whatever/prefix/you/choose/
        archive/
            egcs-1.1b.tar.gz
            binutils-2.9.1.tar.gz
            newlib-1.8.0.tar.gz
            rtems-4.0.0.tgz
            c_build_scripts-4.0.0.tgz
            egcs-1.1b-rtems-diff-19981027.gz
            binutils-2.9.1-rtems-diff-19981027.gz
            newlib-1.8.0-rtems-diff-19981027.gz
            hello_world_c.tgz
            bit
        tools/
            binutils-2.9.1/
            egcs-1.1b/
            newlib-1.8.0/
            rtems-4.0.0/
            bit
            bit_gdb
            bit_rtems
            common.sh
            user.cfg
```

3.4 Host Specific Notes

3.4.1 Solaris 2.x

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 change the first line of the files `bit`, `bit_gdb`, and `bit_rtems` to use the `/bin/ksh` shell instead.

3.4.2 Linux

3.4.2.1 Broken install Program

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 fileutile version 3.16 to resolve this problem.

3.5 Reading the Tools 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.

3.6 Apply RTEMS Patch to EGCS

Apply the patch using the following command sequence:

```
cd tools/egcs-1.1b
zcat ../../archive/egcs-1.1b-rtems-diff-19981027.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/egcs-1.1b
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

3.7 Apply RTEMS Patch to binutils

Apply the patch using the following command sequence:

```
cd tools/binutils-2.9.1
zcat ../../archive/binutils-2.9.1-rtems-diff-19981027.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/binutils-2.9.1
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

3.8 Apply RTEMS Patch to newlib

Apply the patch using the following command sequence:

```
cd tools/newlib-1.8.0
zcat ../../archive/newlib-1.8.0-rtems-diff-19981027.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/newlib-1.8.0
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

3.9 Localizing the Configuration

Edit the `user.cfg` file to alter the settings of various variables which are used to tailor the build process. Each of the variables set in `user.cfg` may be modified as described below:

`INSTALL_POINT`
: is the location where you wish the GNU C/C++ cross compilation tools for RTEMS to be built. It is recommended that the directory chosen to receive these tools be named so that it is clear from which egcs distribution it was generated and for which target system the tools are to produce code for.

  WARNING: The `INSTALL_POINT` should not be a subdirectory under the build directory. The build directory will be removed automatically upon successful completion of the build procedure.

`BINUTILS`
: is the directory under tools that contains binutils-2.9.1. For example:

```
BINUTILS=binutils-2.9.1
```

`GCC`
: is the directory under tools that contains egcs-1.1b. For example,

```
GCC=egcs-1.1b
```

`NEWLIB`
: is the directory under tools that contains newlib-1.8.0. For example:

```
NEWLIB=newlib-1.8.0
```

`BUILD_DOCS`
: is set to "yes" if you want to install documentation. For example:

```
BUILD_DOCS=yes
```

`BUILD_OTHER_LANGUAGES`
: is set to "yes" if you want to build languages other than C and C++. At the current time, this enables Fortan and Objective-C. For example:

```
BUILD_OTHER_LANGUAGES=yes
```

  NOTE: Based upon the version of the compiler being used, it may not be possible to build languages other than C and C++ cross. In many cases, the language run-time support libraries are not "multilib'ed". Thus the executable code in these libraries will be for the default compiler settings and not necessarily be correct for your CPU model.

`RTEMS`
: is the directory under tools that contails rtems-4.0.0.

`ENABLE_RTEMS_POSIX`
: is set to "yes" if you want to enable the RTEMS POSIX API support. At this time, this feature is not supported by the UNIX ports of RTEMS and is forced to "no" for those targets. This corresponds to the `configure` option `--enable-posix`.

`ENABLE_RTEMS_TESTS`
is set to "yes" if you want to build the RTEMS Test Suite. If this is set to "no", then only the Sample Tests will be built. This corresponds to the `configure` option `--enable-tests`.

`ENABLE_RTEMS_TCPIP`
is set to "yes" if you want to build the RTEMS TCP/IP Stack. If a particular BSP does not support TCP/IP, then this feature is automatically disabled. This corresponds to the `configure` option `--enable-tcpip`.

`ENABLE_RTEMS_CXX`
is set to "yes" if you want to build the RTEMS C++ support including the C++ Wrapper for the Classic API. This corresponds to the `configure` option `--enable-cxx`.

3.10 Running the bit Script

After the `bit` script has been modified to reflect the local installation, the modified `bit` script is run using the following sequence:

```
cd tools
./bit <target configuration>
```

Where <target configuration> is one of the following:

- hppa1.1
- i386
- i386-elf
- i386-go32
- i960
- m68k
- mips64orion
- powerpc
- sh
- sparc

If no errors are encountered, the `bit` script will conclude by printing messages similar to the following:

```
The src and build-i386-tools subdirectory may now be removed.

Started:  Fri Apr 10 10:14:07 CDT 1998
Finished: Fri Apr 10 12:01:33 CDT 1998
```

If the `bit` script successfully completes, then the GNU C/C++ cross compilation tools are installed.

If the `bit` script does not successfully complete, then investigation will be required to determine the source of the error.

4 Building RTEMS

4.1 Unpack the RTEMS Source

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

```
cd tools
tar xzf ../archive/rtems-4.0.0.tgz
```

4.2 Add <INSTALL_POINT>/bin to Executable PATH

In order to compile RTEMS, you must have the cross compilation toolset in your search patch. The following command appends the directory where the tools were installed in the previous chapter:

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

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.

4.3 Verifying the Operation of the Cross Toolset

In order to insure that the cross-compiler is invoking the correct subprograms (like `as` and `ld`), one can test assemble a small program. When in verbose more, `gcc` prints out information showing where it found the subprograms it invokes. 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-rtems-gcc -v -S f.c
```

Where `m68k-rtems-gcc` should be changed to match the installed name 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>.

NOTE: One of the most common installation errors is for the cross-compiler not to be able to find the cross assembler and default to using the native `as`. This can result in very confusing error messages.

4.4 Generate RTEMS for a Specific Target and BSP

4.4.1 Using the bit_rtems Script

The simplest way to build RTEMS is to use the `bit_rtems` script. This script interprets the settings in the `user.cfg` file to enable or disable the various RTEMS options.

This script is invoked as follows:

```
./bit_rtems CPU [BSP]
```

Where CPU is one of the RTEMS supported CPU families from the following list:

- hppa1.1
- i386
- i386-elf
- i386-go32
- i960
- m68k
- mips64orion
- powerpc
- sh
- sparc

BSP is a supported BSP for the selected CPU family. The list of supported BSPs may be found in the file tools/rtems-4.0.0/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.

4.4.2 Using the RTEMS configure Script Directly

Make a build directory under tools and build the RTEMS product in this directory. The ../rtems-4.0.0/configure command has numerous command line arguments. These arguments are discussed in detail in documentation that comes with the RTEMS distribution. In the installation described in the section "Unpack the RTEMS source", these configuration options can be found in the file tools/rtems-4.0.0/README.configure.

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-4.0.0/configure --target=<TARGET_CONFIGURATION> \
    --disable-posix --disable-tcpip --disable-cxx \
    --enable-rtemsbsp=<BOARD_SUPPORT_PACKAGE>\
    --prefix=<INSTALL_POINT>
gmake all install
```

Where the list of currently supported of <TARGET_CONFIGURATION>'s and <BOARD_SUPPORT_PACKAGE>'s can be found in tools/rtems-4.0.0/README.configure.

<INSTALL_POINT> is the installation point from the previous step "Modify the bit script" in the build of the tools.

5 Building the Sample Application

5.1 Unpack the Sample Application

Use the following command to unarchive the sample application:

```
cd tools
tar xzf ../archive/hello_world_c.tgz
```

5.2 Set the Environment Variable RTEMS_MAKEFILE_PATH

It must point to the appropriate directory containing RTEMS build for our target and board support package combination.

```
export RTEMS_MAKEFILE_PATH = \
<INSTALLATION_POINT>/rtems/<BOARD_SUPPORT_PACKAGE>
```

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

5.3 Build the Sample Application

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

```
cd tools/hello_world_c
gmake
```

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

5.4 Application Executable

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

```
tools/hello_world_c/o-<BOARD_SUPPORT_PACKAGE>/<filename>.exe
```

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

6 Building the GNU Debugger

GDB is not currently RTEMS aware. The following configurations have been successfully used with RTEMS applications:

- Sparc Instruction Simulator (SIS)
- PowerPC Instruction Simulator (PSIM)
- DINK32

Other configurations of gdb have successfully been used by RTEMS users but are not documented here.

6.1 Unarchive the gdb Distribution

Use the following commands to unarchive the gdb distribution:

```
cd tools
tar xzf ../archive/gdb-4.17.tar.gz
```

The directory gdb-4.17 is created under the tools directory.

6.2 Apply RTEMS Patch to GDB

Apply the patch using the following command sequence:

```
cd tools/gdb-4.17
zcat archive/gdb-4.17-rtems-diff-19981027.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/gdb-4.17
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file.

To see the files that have been modified use the sequence:

```
cd tools/gdb-4.17
find . -name "*.orig" -print
```

The files that are found, have been modified by the patch file.

6.3 Using the bit_gdb script

The simplest way to build gdb for RTEMS is to use the `bit_gdb` script. This script interprets the settings in the `user.cfg` file to produce the gdb configuration most appropriate for the target CPU.

This script is invoked as follows:

```
./bit_gdb CPU
```

Where CPU is one of the RTEMS supported CPU families from the following list:

- hppa1.1
- i386
- i386-elf
- i386-go32
- i960
- m68k
- mips64orion
- powerpc
- sh
- sparc

If gdb supports a CPU instruction simulator for this configuration, then it is included in the build.

6.4 Using the gdb configure Script Directly

6.4.1 GDB with Sparc Instruction Simulation (SIS)

Make the Build Directory

Create a build directory for the SIS Debugger

```
cd tools
mkdir build-sis
```

Configure for the Build

Configure the GNU Debugger for the Sparc Instruction Simulator (SIS):

```
cd tools/build-sis
../gdb-4.17/configure --target-sparc-erc32-aout \
    --program-prefix=sparc-rtems- \
    --disable-gdbtk \
    --enable-targets=all \
    --prefix=<INSTALL_POINT_FOR_SIS>
```

Where <INSTALL_POINT_FOR_SIS> is a unique location where the gdb with SIS will be created.

Make the Debugger

From tools/build-sis execute the following command sequence:

```
gmake all install
```

6.4.2 GDB with PowerPC Instruction Simulator

Make the Build Directory

Create a build directory for the SIS Debugger

```
cd tools
mkdir build-ppc
```

Configure for the Build

Configure the GNU Debugger for the PowerPC Instruction Simulator (PSIM):

```
cd tools/build-ppc
../gdb-4.17/configure \
      --target=powerpc-unknown-eabi \
      --program-prefix=powerpc-rtems- \
      --enable-sim-powerpc \
      --enable-sim-timebase \
      --enable-sim-inline \
      --enable-sim-hardware \
      --enable-targets=all \
      --prefix=<INSTALL_POINT_FOR_PPC>
```

Where <INSTALL_POINT_FOR_PPC> is a unique location where the gdb with PSIM will be created.

Make the Debugger

From tools/build-ppc execute the following command sequence:

```
gmake all install
```

6.4.3 GDB for DINK32

Make the Build Directory

Create a build directory for the DINK32 Debugger

```
cd tools
mkdir build-dink32
```

Configure for the Build

Configure the GNU Debugger to communicate with the DINK32 ROM monitor:

```
cd tools/build-dink32
../gdb-4.17/configure --target-powerpc-elf \
    --program-prefix=powerpc-rtems- \
    --enable-targets=all \
    --prefix=<INSTALL_POINT_FOR_DINK32>
```

Where <INSTALL_POINT_FOR_DINK32> is a unique location where the gdb Dink32 will be created.

Make the Debugger

From tools/build-dink32 execute the following command sequence:

```
gmake all install
```

7 Using MS-Windows as a Development Host

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

This chapter is based on a draft provided by Geoffroy Montel <g_montel@yahoo.com>. Geoffroy's procedure was based on information from David Fiddes <D.J@fiddes.surfaid.org>. Their input and feedback is greatly appreciated.

STATUS: This chapter should be considered preliminary. Please be careful when following these instructions.

7.1 Version Information

This installation process works well under Windows NT. Using Windows 95 or 98 is not recommended although it should be possible with version 3.77 of gmake and an updated cygwinb19.dll.

This procedure should also work with newer version of the tool versions listed in this chapter, but this has not been verified. If you have success with a particular version of the toolset or notice problems in this chapter, please let the RTEMS maintainers know so they can be addressed in future revisions of this document.

7.2 MS-Windows Host Specific Requirements

This section details the components required to install and build a Windows hosted GNU cross development toolset.

7.2.1 Unzipping Archives

You will have to uncompress many 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 Cygwin32.

7.2.2 Text Editor

You absolutely have to use a text editor which can save files with Unix format (so don't use Notepad nor Wordpad). If you do not have an appropriate text editor, try **Programmers File Editor**, it is free and very convenient. This editor may be downloaded from:

http://www.lancs.ac.uk/people/cpaap/pfe/

7.2.3 Bug in Patch Utility

There is a bug in the `patch` utility provided in Cygwin32 B19. The files modified end up having MS-DOS style line termination. They must have Unix format, so a `dos2unix`-like command must be used to put them back into Unix format as shown below:

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

The dos2unix utility may be downloaded from:

ftp://ftp.micros.hensa.ac.uk/platforms/ibm-pc/ms-dos/simtelnet/txtutl/dos2unix.zip

You **must** change the format of every patched file for the toolset build to work correctly.

7.2.4 Files Needed

This section lists the files required to build and install a Windows hosted GNU cross development toolset and their home WWW site. In addition to the sources required for the cross environment listed earlier in Section 3.2 [Get All the Pieces], page 7, you will need to download the following files from their respective sites using your favorite Web browser or ftp client.

cdk.exe http://www.cygnus.com/misc/gnu-win32/

coolview.tar.gz http://www.lexa.ru/sos/

7.2.5 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. The recommended build system spec is:

1. An AMD K6-300, Pentium II-300 or better processor. GNU C and Cygwin32 are **very** CPU hungry.
2. At least 64MB of RAM.
3. At least 400MB of FAT16 disk space or 250MB if you have an NTFS partition.

Even with this spec of machine expect the full suite to take over 2 hours to build with a further half an hour for RTEMS itself.

7.3 Installing Cygwin32 B19

This section describes the process of installing the version B19 of the Cygwin32 environment. It assumes that this toolset is installed in a directory referred to as `<RTOS>`.

1. Execute cdk.exe. These instructions assume that you install Cygwin32 under the <RTOS>\cygnus\b19 directory.
2. Execute Cygwin.bat (either on the start menu or under <RTOS>\cygnus\b19).

3. At this point, you are at the command line of bash, a Unix-like shell. You have to mount the "/" directory. Type:

   ```
   umount /
   mount -b <RTOS> /
   ```

   For example, the following sequence mounts the E:\unix as the root directory for the Cygwin32 environment. Note the use of two \s when specifying DOS paths in bash:

   ```
   umount /
   mount -b e:\\unix /
   ```

4. Create the /bin, /tmp, /source and /build directories.

   ```
   mkdir /bin
   mkdir /tmp
   mkdir /source
   mkdir /build
   mkdir /build/binutils
   mkdir /build/egcs
   ```

5. The light Bourne shell provided with Cygwin B19 is buggy. You should copy it to a fake name and copy bash.exe to sh.exe:

   ```
   cd <RTOS>/cygnus/b19/H-i386-cygwin32/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:

   ```
   cp <RTOS>/cygnus/b19/H-i386-cygwin32/bin/sh.exe /bin
   cp <RTOS>/cygnus/b19/H-i386-cygwin32/bin/bash.exe /bin
   ```

6. Open the file /cygnus/b19/H-i386-cygwin32/lib/gcc-lib/i386-cygwin32/2.7-b19/specs, and change the following line:

   ```
   -lcygwin %{mwindows:-luser32 -lgdi32 -lcomdlg32} -lkernel32
   ```

   to:

   ```
   -lcygwin %{mwindows:-luser32 -lgdi32 -lcomdlg32} -lkernel32 -ladvapi32
   ```

At this point, you have a native installation of Cygwin32 and are ready to proceed to building a cross-compiler.

7.4 Installing binutils

1. Unarchive binutils-2.9.1.tar.gz following the instructions in Section 3.3 [Unarchiving the Tools], page 8 into the /source directory. Apply the appropriate RTEMS specific patch as detailed in Section 3.7 [Apply RTEMS Patch to binutils], page 10.
2. In the /build/binutils directory, execute the following command to configure binutils 2.9.1:

   ```
   /source/binutils-2.9.1/configure --verbose --target=m68k-rtems \
       --prefix=/gcc-m68k-rtems --with-gnu-as --with-gnu-ld
   ```

Replace `m68k-rtems` with the target configuration of your choice. See Section 3.10 [Running the bit Script], page 12 for a list of the targets available.

3. Execute the following command to compile the toolset:

   ```
   make
   ```

4. Install the full package with the following command:

   ```
   make -k install
   ```

   There is a problem with the gnu info package which will cause an error during installation. Telling make to keep going with -k allows the install to complete.

5. In the `cygnus.bat` file, add the directory containing the cross-compiler executables to your search path by inserting the following line:

   ```
   PATH=E:\unix\gcc-m68k-rtems\bin;%PATH%
   ```

6. You can erase the /build/binutils directory content if disk space is tight.
7. Exit bash and run `cygnus.bat` to restart the Cygwin32 environment with the new path.

7.5 Installing EGCS AND NEWLIB

1. Unarchive and patch {No value for "EGCS-TAR"} and newlib-1.8.0.tar.gz following the instructions in Section 3.3 [Unarchiving the Tools], page 8. Apply the appropriate RTEMS specific patches as detailed in Section 3.6 [Apply RTEMS Patch to EGCS], page 10 and Section 3.8 [Apply RTEMS Patch to newlib], page 10.

   NOTE: See Section 7.2.3 [Bug in Patch Utility], page 26.

2. Remove the following directories (we cannot use Fortran or Objective-C as Cygwin32 cross-compilers):

   ```
   /source/egcs-1.1b/libf2c
   /source/egcs-1.1b/gcc/objc
   /source/egcs-1.1b/gcc/f
   ```

   NOTE: See Section 7.2.3 [Bug in Patch Utility], page 26.

3. Link the following directories from Newlib to the main EGCS directory, /source/egcs-1.1b/ :
   - ln -s ../newlib-1.8.0/newlib newlib
   - ln -s ../newlib-1.8.0/libgloss libgloss
4. Change to the /build/egcs directory to configure the compiler:

   ```
   /source/egcs-1.1b/configure --verbose --target=m68k-rtems \
       --prefix=/gcc-m68k --with-gnu-as --with-gnu-ld \
       --with-newlib
   ```

   Replace `m68k-rtems` with the target configuration of your choice. See Section 3.10 [Running the bit Script], page 12 for a list of the targets available.

5. Compile the toolset as follows:

```
make cross
```

You must do a `make cross` (not a simple `make`) to insure that the different packages are built in the correct order. Making the compiler can take several hours even on fairly fast machines, beware.

6. Install with the following command:

   ```
   make -k install
   ```

7. Just as with binutils package, a problem with the gnu info package not building correctly requires that you use -k to keep going.

   ```
   make -k install
   ```

With any luck, at this point you having a working cross-compiler. So as Geoffroy said:

That's it! Celebrate!

Table of Contents