

## Getting Started:

First you need to download the GNU tools. The tools you will need are:

- binutils-2.18
- gcc-3.4.6
- gdb-6.5
- newlib-1.15.0

Feel free to try newer versions of the tools, but these version numbers I am using are known to work, and there were some issues with the version 4 of gcc. Those issues may be fixed in later versions, but I haven't confirmed that.

I'm working on MacOS but these instructions should work for Linux as well. What we will be building are called cross-compilation tools. The compiler and library will be able to compile source code for a different architecture (ARM in this case) than the machine that you are running on.

```
-> cd $HOME
-> mkdir gnutools
-> cd gnutools
-> mkdir build
-> cd build
-> ftp ftp.gnu.org
      (NOTE: please check the list of mirrors for alternate
           download sites)
ftp> cd pub/gnu/gcc/gcc-3.4.6
ftp> binary
ftp> get gcc-3.4.6.tar.bz2
ftp> cd ../../binutils
ftp> get binutils-2.18.tar.bz2
ftp> cd ../gdb
ftp> get binutils-2.18.tar.bz2
ftp> quit
```

Newlib is not maintained by the GNU organization, you will need to download that separately from here:

```
-> ftp sources.redhat.com
ftp> cd pub/newlib
ftp> get newlib-1.15.0.tar.gz
ftp> quit
```

After you have finished retrieving these files its time to start compiling the tools. The first tool that needs to be built is binutils. binutils is a collection of many tools, including the gas (GNU assembler), linker, disassembler and many other useful tools.

```
-> tar xjvf binutils-2.18.tar.bz2
-> cd binutils-2.18
-> ./configure --prefix=${HOME}/gnutools --with-newlib --target=arm-elf
-> make
-> make install
```

Next up is building the GNU C compiler (gcc). Because the building of gcc depends on the GNU assembler, which was built in the previous step, the path to the GNU assembler has to be on your PATH. The first step here is to update your PATH variable with the path to where the GNU tools are installed.

```
(If you are using BASH)
-> export PATH=${PATH}:${HOME}/gnutools/bin

(If you are using CSH or TCSH)
-> setenv PATH ${PATH}:${HOME}/gnutools/bin

-> cd ..
-> tar xjvf gcc-3.4.6.tar.bz2
-> cd gcc-3.4.6
-> ./configure --prefix=${HOME}/gnutools --with-newlib --target=arm-elf --enable-languages=c
-> make
-> make install
```

At this point installed on your system you will have a working compiler, assembler and linker. This is enough to build very simple applications, however you will be missing the standard C library. This library contains implementations of functions like: printf, malloc, free, strcmp, strlen, etc. The next step is to build a C library, I'm using newlib and the instructions are for newlib. If you have a different favorite C library that you like feel free to use it here, but remember, you are on your own.

Building the C library requires the GNU C compiler and the other tools that have already been built. The PATH variable needs to contain the path to these tools. See the instructions above to make sure that your PATH variable is up to date.

```
-> cd ..
-> tar xzvf newlib-1.15.0.tar.gz
-> cd newlib-1.15.0
-> ./configure --prefix=${HOME}/gnutools --with-newlib --target=arm-elf
-> make
-> make install
```

At this point we have everything that is required to build a simple application, but no way to run it. The GNU debugger (gdb) comes with an ARM simulator (along with simulators for many other architectures). The next step is to build the GNU debugger for the ARM architecture.

```
-> cd ..
-> tar xjvf gdb-6.5.tar.bz2
-> cd gdb-6.5
-> ./configure --prefix=${HOME}/gnutools --target=arm-elf
-> make
-> make install
```

### Building your first test application:

```
-> mkdir ~/armsrc
-> cd ~/armsrc
-> vi (or use whatever your favorite editor is) hello.c and insert the
following test
```

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

### To run this program under the GNU debugger:

```
-> arm-elf-gcc -g hello.c
-> arm-elf-gdb a.out
GNU gdb 6.5
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for
details.
This GDB was configured as "--host=i386-apple-darwin8.10.3 --
target=arm-elf"...
(gdb) target sim
Connected to the simulator.
(gdb) break main
Breakpoint 1 at 0x8228: file hello.c, line 5.
(gdb) load
Loading section .init, size 0x1c vma 0x8000
Loading section .text, size 0x85e8 vma 0x801c
Loading section .fini, size 0x18 vma 0x10604
Loading section .rodata, size 0x234 vma 0x1061c
Loading section .eh_frame, size 0x4 vma 0x18850
Loading section .ctors, size 0x8 vma 0x18854
Loading section .dtors, size 0x8 vma 0x1885c
Loading section .jcr, size 0x4 vma 0x18864
```

```
Loading section .data, size 0x940 vma 0x18868
Start address 0x8110
Transfer rate: 298304 bits in <1 sec.
(gdb) r
Starting program: /Users/jnewlin/armsrc/a.out

Breakpoint 1, main () at hello.c:5
5         printf("Hello World!\n");
(gdb) n
Hello World!
6         return 0;
(gdb)
```

## Patching the ARM Simulator

The ARM architecture (ARMv4) does not have a timer as part of the processor core, and hence the ARM simulator that comes with the GNU debugger does not simulate a timer. However for an RTOS to work, you must be able to have a regular timer tick event. The timer tick is primarily for task switching, however it can be used for other things, for example implementing a sleep() type function or implementing timeouts for various system calls.

- Download the patch file
- cd \$HOME/gnutools/build
- patch -p1 < gdb-6.5.patch

The output should look like:

```
-> patch -p1 < gdb-6.5/gdb-6.5.patch
patching file gdb-6.5/sim/arm/armcopro.c
patching file gdb-6.5/sim/arm/armsupp.c
patching file gdb-6.5/sim/arm/armvirt.c
patching file gdb-6.5/sim/arm/wrapper.c
```

Now cd to gdb-6.5 and run `make` and `make install` again.

## Building DrRtos

- Download the latest DrRtos source code
- unpack the source
  - tar xzf drrtos.tar.gz
- export MAKE\_CONFIG=Makefile.config.arm
- make

One sample application will be built, and show up in the bin directory. The name of the test program is called `simple`. This test creates three different threads that each take a mutex, and call `printf` with a message.

To run the test simply type:

```
-> arm-elf-run simple
```

You can use the GNU debugger to debug the test application.

```
-> arm-elf-gdb simple
(gdb) target sim
Connected to the simulator.
(gdb) load
Loading section .init, size 0x1c vma 0x8000
Loading section .text, size 0x9894 vma 0x8020
Loading section vectors, size 0x2c vma 0x118b4
Loading section .fini, size 0x18 vma 0x118e0
Loading section .rodata, size 0x254 vma 0x118f8
Loading section .eh_frame, size 0x4 vma 0x19b4c
Loading section .ctors, size 0x8 vma 0x19b50
Loading section .dtors, size 0x8 vma 0x19b58
Loading section .jcr, size 0x4 vma 0x19b60
Loading section .data, size 0x968 vma 0x19b64
Start address 0x8114
Transfer rate: 337472 bits in <1 sec.
(gdb) break ThreadOne
(gdb) run
(gdb) c
Continuing.

Breakpoint 1, ThreadOne (arg=0x0) at simple.c:33
33      dr_rtos_mutex_init(&print_mut);
(gdb)
```

And so on.