Back

GBA Dev In Linux

GCC for GBA for Linux

These are step by step instructions for building a gcc cross compiler for the gba for linux. I typed them as I installed them on my second machine. It is an Athlon 900mhz, 256MB RAM, 6gig test partition with fresh Redhat 9 installed (only updates and nVidia X11/OpenGL drivers installed), nVidia Gefore2 MX 32mb, SB Live, 3Com 3c905b NIC, other generic cdrom/floppy.... those stats might be usefull since I will time all of the big compiles.

Step 0: Get A Working Linux System

Your by yourself on this one,**Good Luck!**

Step 1: Get The Sources

Program	What version I used What is in it	Links
Binutils	binutils-2.11.2.tar.bz2 assembler, linker, objcopy, other goodies... (versions newer than this create overlap errors with crt0.o?) (some of them complain about --mcpu=arm7tdmi with certain versions of gcc! I am experimenting)	Binutils From ftp.gnu.org Mirror:Binutils From mirrors.kernel.org
GCC	gcc-3.0.4.tar.gz c/c++ compiler (will experiment with newer ones but gcc-3.3 gave me trouble though all I think it was doing was passing --mcpu=arm7tdmi to as!)	GCC From ftp.gnu.org Mirror:GCC From mirrors.kernel.org
Newlib	newlib-1.11.0.tar.gz micro libc	Newlib From sources.redhat.com Mirror: still looking...
crtls v1.28	crtls.zip start of rom/mb image.	crtls.zip from www.devrs.com Mirror: crtls.zip from

a mirror

Step 2: Properly (tar/bunzip2/gunzip)ing the source

What is happening	Command Line	Time It Took
Do this this way or you may have troubles!		
We have to build binutils/gcc/newlib in a seperate directory from the source! so we create 3 directories.	**mkdir build-binutils** **mkdir build-gcc** **mkdir build-newlib**	how fast can you type?
Now in each of those directories, uncompress the related source I will just give you one example you can do the rest	(in build-binutils) **tar xfvj ../binutils-2.11.2.tar.gz** (your path to binutils-2.11.2.tar.gz may be different)	not to long

Step 3: Building Build Tools

subStep 3.1: Building Binutils

What is happening	Command Line	Time It Took
Now we need to run configure inside the binutils directory, from our *build-utils* directory with a few options **--target=arm-thumb-elf** which means build for arm (uh yeah) **--prefix=/somedir** i don't use this but it allows you to install the files in a directory other than the default /usr/local	**./binutils-2.11.2/configure --target=arm-thumb-elf**	3 seconds
Now we start the build, still in the *build-binutils* directory we created.	**make**	5 minutes 17 seconds
Now we install the files to whatever you set --prefix to or /usr/local if you didn't use --prefix (still in the *build-binutils* directory we created).	**make install**	35 seconds

subStep 3.2: Building GCC

What is happening	Command Line	Time It Took
Now we need to run configure inside the gcc directory, from the *build-gcc* directory with a few options **--target=arm-thumb-elf** arm/thumb output assembly **--with-cpu=arm7tdmi** default processor	**./gcc-3.0.4/configure --target=arm-thumb-elf --with-cpu=arm7tdmi --with-newlib --enable-multilib --enable-interwork**	1 minute

type (there are alot of other ARMs) **--with-newlib** use newlib instead of glibc **--enable-multilib** not sure, I think it is to help with interworking **--enable-interwork** make arm and thumb play nice together **--disable-threads** don't use threads? **--enable-targets=arm-elf** use elf format for objects **--with-headers=../build-newlib/newlib-1.11.0/newlib/libc/include/** use headers from our freshly decompressed newlib (may have to change path) **--enable-languages="c"** just c no c++ or ada or whatever else gcc does... **--prefix=/somedir** i don't use this but it allows you to install the files in a directory other than the default /usr/local	**--disable-threads --enable-targets=arm-elf --with-headers=../build-newlib/newlib-1.11.0/newlib/libc/include --enable-languages="c"** (your newlib path may be different) (that is one line if you didn't now)	
Now we start the build, still in the *build-gcc* directory we created.	**make**	7 minutes 52 seconds (I thought it would take longer....)
Now we install the files to whatever you set --prefix to or /usr/local if you didn't use --prefix (still in the *build* directory we created).	**make install**	27 seconds (gcc installs faster then binutils?!)

subStep 3.3: Building Newlib

What is happening	Command Line	Time It Took
Now we need to run configure inside the newlib directory, from our new *build-newlib* directory with a few options **--target=arm-thumb-elf** which means build for arm (uh yeah) **--prefix=/somedir** i don't use this but it allows you to install the files in a directory other than the default /usr/local	**./newlib-1.11.0/configure --target=arm-thumb-elf**	5 seconds
Now we start the build, still in the *build-newlib* directory we created.	**make**	5 minutes 46 seconds
Now we install the files to whatever you set --prefix to or /usr/local if you didn't use --prefix (still in the *build* directory we created).	**make install**	1 minute 21 seconds

subStep 3.4: Building crtls

What is happening	Command Line	Time It Took

After you have unzip'ed the crtls.zip file just run it through our new assembler.	**arm-thumb-elf-as CRT0.S -o crt0.o**	~0 seconds
You might want to get rid of these to. (yes you do) /usr/local/arm-thumb-elf/lib/crt0.o /usr/local/arm-thumb-elf/lib/redboot-crt0.o /usr/local/arm-thumb-elf/lib/thumb/crt0.o /usr/local/arm-thumb-elf/lib/thumb/redboot-crt0.o		how fast can you type?

Step 4: Building Send Tools

subStep 4.1: Building mb (mbv2)

After you have downloaded and decompressed mblinux.tar.gz you, like me, have just found out that you don't actually need to build it... (I forgot)

subStep 4.2: Building fl (flash advance)

Command	What it does...	Time It Took
After you have downloaded and decompressed flgba.zip compiling is simple	**gcc fl.c -o fl** (that is x86 gcc not arm-thumb-elf-gcc) (I copy fl and mb to /usr/local/bin so that they are in path)	~1 second

Step 5: Testing it all

Here is a small program with makefile I wrote to test the build

First you need to copy that crt0.o you created earlier, and lnkscript from crtls.zip (same place you got CRT0.S from) to the directory where test.c and it's makefile are then you run this incredibly complex command	**make** - for a .mb image or **make all** - for a .mb and .gba image	~1 second for make all
then to send it over an mbv2 cable	**make send** or **mb -w 10 -s test.mb** (only if you have mb in your path though!)	2-3 seconds, I forgot to time it.

If you see a single white dot in the middle of gba screen you are finished

Go try my demos/PSUEDO tutorial if you are brave or go to www.gbadev.org for some other tutorials

Credits/Other Reading

Who Where	Note
Dooby His site	How I learned how to build GCC for GBA for Linux **Thanks** Dooby!

?? Cross GCC Howto	Helped alittle... kind of old I think
Jason Wilkins devkitadv source	mostly just confused me, strange buildscript/makefile (can't remember if it was sh or make)... Read this...

Back

This page was created using EMACS and The GIMP. It was tested with Mozilla
And if you where wondering where step 2.0, 3.0 or any other .0 it was 2=2.0,3=3.0....!
And and ^K ^Y made %30 percent of this file!