Tom’s Networking

How To: Hacking the Linksys NSLU2 – Part 3 Adding an iTunes server

By Jim Buzbee – August 24 2004

Introduction

If you've been following along with my previous articles [Part 1] [Part 2], you now have the ability to telnet into your NSLU2 and start up NFS. Those are valuable capabilities, but this time we're going for the entertainment value.

Like many people, my MP3 collection is beginning to eat up some serious disk space. As well as getting my music moved to central storage, it would also be nice to share it among different computers in the household. Does the NSLU2 have the right stuff to be a music server? It's small enough to tuck into your entertainment center. It's silent, and its storage capabilities are limited only by the the size of the disk you plug into it. It looks promising to me!

A reader of my previous articles suggested a look at an iTunes server called mt-daapd. Running mt-daapd on the NSLU2 would allow you to store all of your music in one place and serve it up to Mac OSX and Windows machines running iTunes.

An interesting feature of mt-daapd is that it uses Rendezvous to advertise itself to the local network. This means automatic discovery of our NSLU2 music server. Fire up iTunes on either a Windows or a Mac OSX machine and our little server automatically shows up as an available music library. Very cool. Building mt-daapd will be a bit more challenging than our previous NFS build, but step-by-step I'll show how to do it.

> ⚠ Disclaimer: It goes without saying, but I'll say it anyway. Exploring the NSLU2 by looking at its internal file structures using any method that doesn't modify its code should leave your warranty intact. But modifying the NSLU2 in any way will void your warranty.
>
> TomsNetworking, Tom's Guides Publishing and I are not responsible for any damage that the information in this article may cause to your NSLU2 or any data it manages.
>
> So download a copy of the current firmware before you start, and don't go trying to get help from Linksys if you break it.

Gathering the sources

I'm assuming that you've followed my previous article and have installed the tool-chain we'll be using to build our server. I'll also be writing this assuming bash syntax. If you use a different shell, you'll have to adjust. As in my last article, I'll be doing the build from a Mandrake machine. But this time I'll be NFS mounting my NSLU2 disk so that my build will automatically end up where I need to run it.

The first step in this process is to create a work area for the packages that we're going to build. I created a directory called iTunes under my home directory on the NSLU2. Next we'll need to get all of the various source packages required. We'll naturally start with the mt-daapd source from here

A glance at the documentation shows that we'll need libid3tag, the library used for ID3 tag reading and gdbm, the GUN database manager. A quick test I did showed that we will also need the zlib compression library. I fetched libid3tag from here, gdbm from here and zlib from here. I placed all of the required packages in my iTunes directory. After un-tarring them, you'll end up with a gdbm-1.8.3 directory, a libid3tag-0.15.1b directory, a mt-daapd-0.2.0 directory and a zlib-1.2.1 directory.

Before we start the build, we'll set up some common environment variables that will be used during the cross-compile. All of the packages in this build are configured using the same mechanism, the GNU Autoconf package. This package can look at specific environment variables in order to tell it where to find the compiler, linker, include files, etc.

I created a setup.sh file with the following definitions :

```
export BASE=/mnt/nslu2/data/jim/iTunes/
export LINUX=$HOME/snapgear/linux-2.4.x/include/
export TOOLSBASE=$HOME/gpl_code_2.03/usr/

     local/openrg/armsaeb/bin/armv4b-hardhat-linux
export CC=$TOOLSBASE-gcc
export STRIP=$TOOLSBASE-strip
export RANLIB=$TOOLSBASE-ranlib
export LDFLAGS=-L$BASE/lib
export CFLAGS="-I$BASE/include/ -I$LINUX"
```

(Note that in the setup.sh file the line starting with "export TOOLSBASE" and the next indented line are entered as one line with no space between them.).

BASE is defined to be the top level of my working area while LINUX points to the snapgear Linux source tree. TOOLSBASE is where my cross-compilation tools live and CC, STRIP and RANLIB end up being the full path name of their executables.

LDFLAGS and CFLAGS will be used during our builds and just point into our build area and the Linux source tree. Create your version of the above file using your specific paths. Then "dot" the file, i.e. . setup.sh so that the variables get into your environment. Now we're ready to do our first build.

Easy builds first

It's important to build our packages in the right order. The most basic package seems like zlib since it shouldn't have any external dependencies. In the zlib directory enter the following command:

```
./configure --prefix=$BASE
```

The prefix argument tells Autoconf where to place the result of the build. When executing the script, you should get a few lines of output and should see your arm-gcc compiler referenced. Now that it's properly configured, just do a simple:

```
make install
```

After a number of files are compiled and a library created, the resultant files are copied into our base directory. That was surprisingly simple. One package down, three more to go.

We'll tackle the gdbm package next. It should also have no external dependencies, but we'll have to tell it a little more about how we want the package configured. In the gdbm directory execute the following command:

```
./configure --host=arm-linux --prefix=$BASE
```

This Autoconf setup requires us to tell it which architecture we are building for and the host argument is used for that. Executing this command causes a much longer configuration listing than our last one, but it should finish up in a few seconds. Next do the build by issuing the command:

```
make install
```

This build is a little bit ugly. You should see a long compilation, followed by ldconfig complaining about the resultant shared library because it doesn't understand arm binaries. We're not concerned with this because we're obviously not going to be using the libraries on this box. You may also see an error regarding changing the ownership of gdbm.h. This error is also harmless to our build.

Now on to libid3tag. From the libid3tag directory:

```
./configure --host=arm-linux --prefix=$BASE
```

This generates a long listing, then as before:

```
make install
```

This build will complete as before with a harmless complaint from ldconfig. Our libraries are now complete. This has been all too easy right? Well now it will get just a little more difficult as we move on to mt-daapd.

Building mt-daapd

As I said, the mt-daapd build is somewhat more involved. But don't worry, we'll get through it.

From the mt-daapd directory, execute the following command:

```
./configure --host=arm-linux --prefix=$BASE

LIBS="-lgdbm -lid3tag -lz" ac_cv_func_setpgrp_void=yes
```

(Note that the above command is entered on one line with a space between $BASE and LIBS.)

This deserves a bit of explanation. The host and the prefix options we've seen before. The LIBS options is specific to this package. For whatever reason, the stock build was listing the libraries before the .o files during the link phase, causing the link to fail with undefined references.

Investigation of the configure script showed an option, LIBS, that could be used to add extra libraries to the link. So I'm using it here to add back in our libraries. It's a hack, but it saves us from having to make modifications to the configure script itself.

The other new option, ac_cv_func_setpgrp_void is also a bit of a hack. When I first ran the configure script I was getting an error from a specific portion that wanted to run a test program to determine the behavior of a particular function. Of course, with cross-compilation we can't compile and run programs locally. By setting this variable at configure time, we're essentially telling the script that we've already run this particular test and have the answer. This required me to manually determine the correct answer to this test and pass it along on the command line.

> ⚠NOTE: From my experience, this is a common problem when doing cross-compilation and using Autoconf. The authors of a package often don't have a chance to build for a different architecture so the configure script is just not quite correct for a cross-compile. This technique, while a bit ugly, will at least allow you to complete the configuration process.

After running the above command, we'll still have to fix a couple of things before we can do our build. First, in order to simplify our installation, we're going to remove the shared libraries that we just built. When the shared libraries are removed, the link will occur against the static versions. This keeps us from having to add additional libraries that no other process is likely to use. If you find other processes that need these libraries, you can skip this step and add the libraries to your system. Execute the following command from the mt-daapd directory:

```
rm -f ../lib/*.so*
```

Next, due to the way that the arm-compiler deals with structures and unions, we'll have to make a code change. By default, the arm compiler pads out structures and unions to 16 bit boundaries. Normally this wouldn't cause a problem, but in our case, structures are read and written directly to the network. The iTunes program on the other side of the network connection doesn't have the padding, resulting in a garbled conversation. The file we'll have to change is src/mDNSClientAPI.h, and the change we'll make is to add an __attribute__ ((packed)) compiler directive to the existing definition.

The following types also need to have the packed attribute added:

- mDNSOpaque16
- mDNSOpaque32
- mDNSOpaque128
- mDNSAddr
- rdataSRV
- rdataMX

After the change, my definition of mDNSOpaque16 looks like this:

```
typedef union { mDNSu8 b[2]; mDNSu16 NotAnInteger; }

__attribute__ ((packed)) mDNSOpaque16;
```

(Again, entered as one line, no space between the } and __ .)

The special compilation attribute falls between the right brace and the typedef name. I passed these changes on to the author of mt-daapd, but the changes may or may not get applied since they are fairly specific to the gcc arm compiler.

After adding these attributes, a "make install" will finish up the build and deposit the result in ../sbin/. Finally strip the resultant binary to reduce its size:

```
$STRIP ../sbin/mt-daapd
```

and we're done! On to the install!

Installing to the NSLU2

Now it's time to install. First we'll modify the mt-daapd.conf file in the contrib directory. The file is well documented, so it should be easy to modify it to suit your needs and specific directory paths. For my directory structure, I created a top-level mt-daapd directory: /share/hdd/data/mt-daapd.

Inside this, I copied the admin-root directory that is used for the web-based configuration and status interface. I created a cache sub-directory for the song database and copied the contrib/mt-daapd.playlist into the mt-daapd directory, modifying the config file accordingly. For my actual music database, I created a top-level MP3 directory and made the configuration file changes to reflect it.

When you've modified the configuration file to suit your needs, copy it to the /share/hdd/conf/etc directory, and create a symbolic link from etc:

```
ln -s /share/hdd/conf/etc/mt-daapd.conf /etc/mt-daapd.conf
```

Now copy the actual binary, src/mt-daapd, to /share/hdd/conf/bin/. Then move over some music files. I found that the server takes MP3's and AAC files and can scan a directory tree rather than just a flat directory of files. I moved my files over using NFS, but depending on where you put your directory, you can also use SMB or just ftp.

As we did with NFS in my last article, you should create an rc script that creates the symbolic link and starts the actual process. I've been using /etc/rc.d/rc.samba as a starting point. Copy it to /share/hdd/conf/rc.d/rc.mt-daapd, modify it for mt-daapd and start it up:

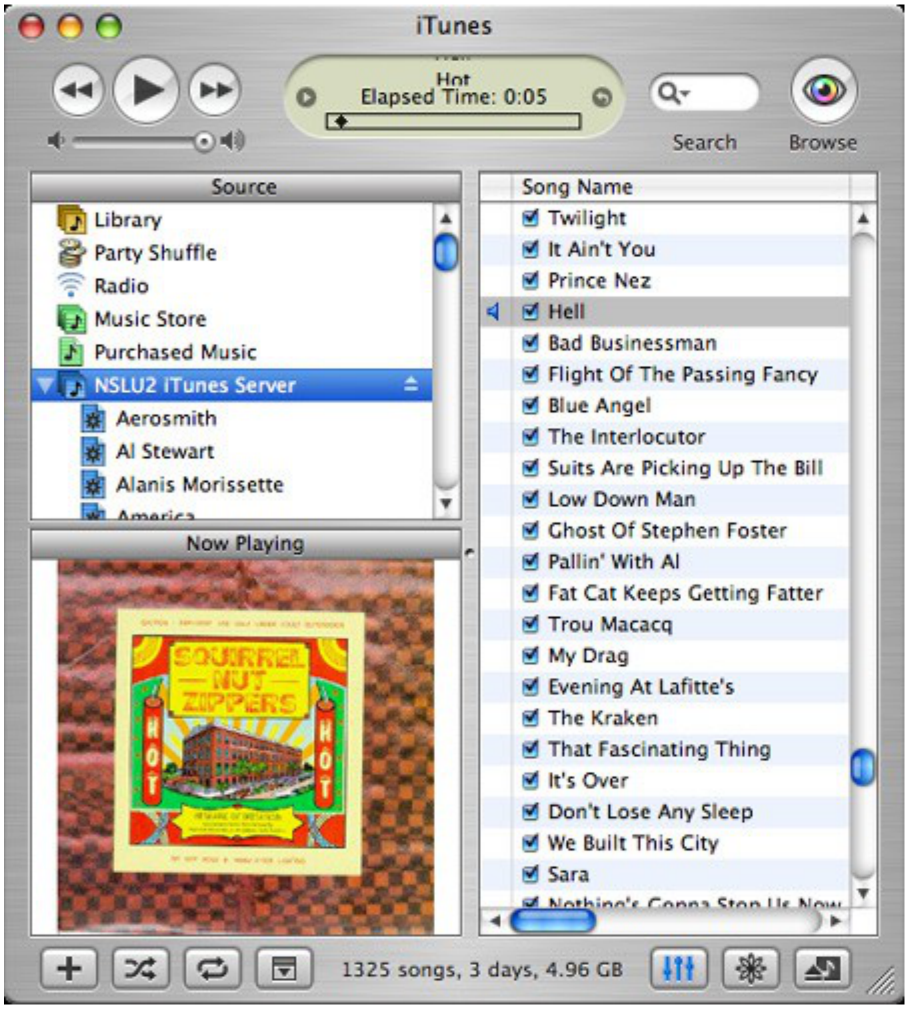
```
/share/hdd/conf/rc.d/rc.mt-daapd
```

A quick check of /var/log/mt-daapd.log should show some status messages and issuing a ps -ax command should show mt-daapd running. Don't be alarmed when you see multiple lines in the psoutput for mt-daapd. The mt stands for multi-threaded and Linux will show each thread separately in the output.

Play that funky music

Of course the final test is streaming music to your PC. This server will support either Windows or Mac OSX versions of iTunes so whichever version you have, start it up. On the left hand side, in the "source" pane of the iTunes window, you should see a little blue icon with the name of your server as specified in the mt-daapd configuration file. Very slick. No searching, no configuration, nothing to do, it just appears.

If you've gotten this far, I shouldn't have to tell you what to do next. Select the blue icon, see your list of music, choose one and play it! I've successfully served music to three wireless laptops simultaneously. This little box has a lot more potential than Linksys gave it credit for!



Twilight
It Ain't You
Prince Nez
Hell
Bad Businessman
Flight Of The Passing Fancy
Blue Angel
The Interlocutor
Suits Are Picking Up The Bill
Low Down Man
Ghost Of Stephen Foster
Pallin' With Al
Fat Cat Keeps Getting Fatter
Trou Macacq
My Drag
Evening At Lafitte's
The Kraken
That Fascinating Thing
It's Over
Don't Lose Any Sleep
We Built This City
Sara
Nothing's Gonna Stop Us Now

1325 songs, 3 days, 4.96 GB

Figure 1 iTunes in action, courtesy of your modified NSLU2

Up to this point, we've limited ourselves to adding new packages to the NSLU2. Next time, prepare to shred your warranty, dig deeper and make some permanent changes. We'll burn our own flash, giving us a hook to automatically start our new processes. In the meantime, you can follow my adventures on my Linux on the NSLU2 page.

Tip: If you want to skip all this stuff and just want a binary that you can load onto your NSLU2, you can download the 185KB file from here.