Subject: Linking standard libraries
Posted by dolik.rce on Sun, 29 Jan 2012 21:03:20 GMT
View Forum Message <> Reply to Message

Hi everyone,

Is there any important reason why we use our own copy of some common standard libraries, such as libjpeg or libbz2 on platforms where they are almost always present as shared libs? I understand that having the copy of code is useful for windows, but in POSIX platforms we can just link against the shared libs, as it is done already in some plugins, e.g. in plugin/png.

I've noticed this while checking the quality of debian packages with lintian tool. It considers using libjpeg and libbz2 statically linked in to be an error. Here is the explanation, for completeness:lintian -i theide_4424-0~squeeze0_amd64.debE: theide: embedded-libjpeg ./usr/bin/theide
N:
N: The given ELF object appears to have been statically linked to libjpeg.
N: Doing this is strongly discouraged due to the extra work needed by the
N: security team to fix all the extra embedded copies or trigger the
N: package rebuilds, as appropriate.
N:
N: If the package uses a modified version of libjpeg it is highly
N: recommended to coordinate with the libjpeg maintainer to include the
N: changes on the system version of the library.
N:
N: Refer to Debian Policy Manual section 4.13 (Convenience copies of code)
N: for details.
N:
N: Severity: serious, Certainty: possible
N:
E: theide: embedded-library ./usr/bin/theide: bzip2
N:
N: The given ELF object appears to have been statically linked to a
N: library. Doing this is strongly discouraged due to the extra work needed
N: by the security team to fix all the extra embedded copies or trigger the
N: package rebuilds, as appropriate.
N:
N: If the package uses a modified version of the given library it is highly
N: recommended to coordinate with the library's maintainer to include the
N: changes on the system version of the library.
N:
N: Refer to Debian Policy Manual section 4.13 (Convenience copies of code)
N: for details.
N:
N: Severity: serious, Certainty: possible
N:

The changes required to get rid of these errors and to make U++ more standards compliant

should be quite minimal. For starters, I attach patch for plugin/bz2. Converting plugin/jpg might be bit trickier, but I'll try to do it soon as well.

I'm just not sure if there is any real reason that would prevent merging these patches to U++?

Best regards,
Honza

File Attachments

1) bz2.patch, downloaded 410 times

Subject: Re: Linking standard libraries
Posted by mirek on Mon, 30 Jan 2012 07:45:07 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Sun, 29 January 2012 16:03Hi everyone,

Is there any important reason why we use our own copy of some common standard libraries, such as libjpeg or libbz2 on platforms where they are almost always present as shared libs?

I guess nothing else than to reduce number of required prerequisites to run U++. In theory, embedding the code covers more platforms than not to.

But I agree that we perhaps should 'move' some libraries to png-like state.

Subject: Re: Linking standard libraries
Posted by dolik.rce on Mon, 30 Jan 2012 08:06:24 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 30 January 2012 08:45I guess nothing else than to reduce number of required prerequisites to run U++. In theory, embedding the code covers more platforms than not to.

But I agree that we perhaps should 'move' some libraries to png-like state.

I wonder if there is some way to figure out if there is given library present at compile time? So that we could switch between the outer linking and embedded code as necessary... OTOH, this would probably lead to having such code in TheIDE and stronger dependency of U++ on this particular IDE, which is not a good thing.

So what about adding a flag into each of these libraries, something like EMBED_JPG? It would be always on for win32 and on other platforms, developer can use it only when he needs it (e.g. when he knows he will ship his app to system where libjpeg is missing)?

Honza

Subject: Re: Linking standard libraries
Posted by mirek on Mon, 30 Jan 2012 09:21:48 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 30 January 2012 03:06mirek wrote on Mon, 30 January 2012 08:45I guess nothing else than to reduce number of required prerequisites to run U++. In theory, embedding the code covers more platforms than not to.

But I agree that we perhaps should 'move' some libraries to png-like state.

I wonder if there is some way to figure out if there is given library present at compile time? So that we could switch between the outer linking and embedded code as necessary... OTOH, this would probably lead to having such code in TheIDE and stronger dependency of U++ on this particular IDE, which is not a good thing.

So what about adding a flag into each of these libraries, something like EMBED_JPG? It would be always on for win32 and on other platforms, developer can use it only when he needs it (e.g. when he knows he will ship his app to system where libjpeg is missing)?

Honza

I guess let us just move them... In the end, we already depend on some shared libraries that are in fact less likely to be present (e.g. gtk), so moving bz2 etc is not a big issue IMO.

That said, do we have some list of libs to move?

Subject: Re: Linking standard libraries
Posted by dolik.rce on Mon, 30 Jan 2012 11:27:07 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 30 January 2012 10:21That said, do we have some list of libs to move?
Great We can start with those that lintian itself believes they should not be linked statically:
jpeg   // should be converted (high priority)
bzip2   // should be converted (high priority, patch above)
pcre3   // should be converted
tiff   // should be converted
zlib   // should be converted
png   //already done
expat   // not used in U++
file   // not used in U++
libxml2   // not used in U++

openjpeg // not used in U++

Honza

Subject: Re: Linking standard libraries
Posted by mirek on Mon, 30 Jan 2012 12:15:32 GMT
View Forum Message <> Reply to Message

zlib is 'already done' too...

Subject: Re: Linking standard libraries
Posted by cbpporter on Mon, 30 Jan 2012 19:58:47 GMT
View Forum Message <> Reply to Message

After the change will be optionally still be able to link statically with U++ local packages?

Subject: Re: Linking standard libraries
Posted by mirek on Mon, 30 Jan 2012 20:01:23 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 30 January 2012 14:58After the change will be optionally still be able to link statically with U++ local packages?

Not sure I understand question. IMO sitation will not change much from today - even today U++ requires a couple of shared libraries (png, zlib, gtk, glib...). We are only going to extend this list.

Subject: Re: Linking standard libraries
Posted by mirek on Tue, 31 Jan 2012 19:11:46 GMT
View Forum Message <> Reply to Message

bz2 patch applied, but unfortunately it is not possible to do the same thing with jpeg - its internal headers lack include guards, and even with them, it is not possible to mix files beacause of #defines.

Maybe the solution here is to add proxy .c files for all jpeg sources...

Subject: Re: Linking standard libraries
Posted by dolik.rce on Tue, 31 Jan 2012 20:50:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 31 January 2012 20:11bz2 patch applied, but unfortunately it is not possible to do the same thing with jpeg - its internal headers lack include guards, and even with them, it is not possible to mix files beacause of #defines.

Maybe the solution here is to add proxy .c files for all jpeg sources...
I like to believe anything is possible I'll try to look at it. And BTW, I've noticed that the libjpeg in U++ is very old. If the README file is correct then 14 years and definitely more than 4 years (based on commit date). I'll try to upgrade it in the process to version 8, preferably to the lijpeg-turbo implementation, which uses SIMD instructions to make everything faster and should be also more memory efficient.

Honza

Subject: Re: Linking standard libraries
Posted by Sender Ghost on Fri, 03 Feb 2012 11:04:42 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 30 January 2012 20:58After the change will be optionally still be able to link statically with U++ local packages?
I guess, no. Like with plugin/png already.

In case you interested in such approach, I made mentioned solution.
It is possible to use .NOEXTLIB main configuration flag to use local U++ packages. Currently, for bzip2, jpeg, tiff, png, zlib libraries.
But such approach requires some maintenance on the files of the libraries (adding "#if flagNOEXTLIB || flagWIN32" to begin and "#endif" to the end of the *.c/cpp files), in case of updating. Also, it will be still possible to explore U++ libraries with current Assist++ (compared to "import by #include" solution).

You could find changed files and diff file (based on 4524 revision) for such solution in the attachment.

Edit: There is also second method.

File Attachments
1) noextlib_r4524_first.zip, downloaded 394 times

Subject: Re: Linking standard libraries
Posted by dolik.rce on Fri, 03 Feb 2012 12:19:54 GMT
View Forum Message <> Reply to Message

Sender Ghost wrote on Fri, 03 February 2012 12:04cbpporter wrote on Mon, 30 January 2012 20:58After the change will be optionally still be able to link statically with U++ local packages?
I guess, no. Like with plugin/png already.

In case you interested in such approach, I made mentioned solution.
It is possible to use .NOEXTLIB main configuration flag to use local U++ packages. Currently, for bz2, jpeg, tiff, png, zlib libraries.I would like to have such flag too, just in case. Even better if it could be controlled on per package basis as well. E.g. .NOEXTLIB would switch all and .NOEXTPNG would use only plugin/png as static. It should be simple to do.

Sender Ghost wrote on Fri, 03 February 2012 12:04But such approach requires some maintenance on the files of the libraries (adding "#if flagNOEXTLIB || flagWIN32" to begin and "#endif" to the end of the *.c/cpp files), in case of updating. Also, it will be still possible to explore U++ libraries with current Assist++ (compared to "import by #include" solution).
Well, the maintenance could be probably automated, using a script, so it is not a big deal. Of course the "import by include" method makes this easier, but as you mentioned it hides the files from Assist (altough I don't remember ever looking into those sources...). There is also third option, having separate package for library sources (example for jpg here), which would take the best of both previous options, for the price of having longer package list...

I think the final decision won't be easy, there is many options and each has its pros and its cons...

Honza

Subject: Re: Linking standard libraries
Posted by Sender Ghost on Fri, 03 Feb 2012 15:22:13 GMT
View Forum Message <> Reply to Message

After private conversation with dolik.rce, I think, there is another solution with additional packages of local libraries. It is possible to use .EXTLIB and/or .NOEXTLIB main configuration flags (with .EXTLIB precedence over .NOEXTLIB) or don't using them for the same build behaviour (basically, local libraries used for Windows operating system and external libraries otherwise).

Currently, for bzip2, jpeg, png and zlib libraries.

You could find changed files and diff file (based on 4524 revision) for such solution in the attachment.

Edit: The tiff library excluded, after testing on Linux operating system, because of wrapper package used internal function(s) of the local library and requires further converting. Also fixed includes for plugin/png package.

File Attachments
1) noextlib_r4524_second.zip, downloaded 405 times

Subject: Re: Linking standard libraries
Posted by dolik.rce on Wed, 18 Jul 2012 19:35:26 GMT
View Forum Message <> Reply to Message

Hi everyone

I know it took me a lot of time, but I finally did it Tonight I committed a new versions of jpg, png and tif plugins into sandbox for testing. I don't have M$ windows, so I only tested with MSC9 and MINGW in wine, so please test on real win32 machines If you agree with what I did so far, I can convert other plugins as well to follow the same scheme.

The main features:

Each plugin has a shell script that can be used to upgrade the package when new upstream version of the library is released. It should require only minor tweaks on each upgrade, thus simplifying things a lot.

On win32, the plugins are always compiled and statically linked. On posix they are by default linked dynamically to system version, but static linking can be forced by using .EMBED_JPG, .EMBED_PNG and/or .EMBED_TIF flags.

Current versions of libraries used are:
  libjpeg-turbo 1.2.0
  tiff 4.0.1
  libpng 1.5.10
(I know they are already out of date, but they were bleeding edge when I first started working on this)

As far as I could test the packages work, the only issue I know about is a memory leak when compiling with MSC, which I didn't locate yet.

Let me know your opinions, test, send patches, etc... Thanks.

Best regards,
Honza

Subject: Re: Linking standard libraries
Posted by mirek on Sun, 22 Jul 2012 16:26:50 GMT
View Forum Message <> Reply to Message

I am afraid that the new code does not have the same error handling as the old one - you can quickly demonstrate that using examples/ImageView, which now does not work.

Mirek

Subject: Re: Linking standard libraries
Posted by dolik.rce on Sun, 22 Jul 2012 20:42:43 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 22 July 2012 18:26I am afraid that the new code does not have the same error handling as the old one - you can quickly demonstrate that using examples/ImageView, which now does not work.

Mirek

Ok, I'll have another look at it. So far I tested with reference/ImageEncoders which seemed to work...

Honza

Subject: Re: Linking standard libraries
Posted by dolik.rce on Tue, 24 Jul 2012 05:13:35 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 22 July 2012 18:26I am afraid that the new code does not have the same error handling as the old one - you can quickly demonstrate that using examples/ImageView, which now does not work.

Mirek

Well, the error handling is working well. The problem seems to be that sizeof(jpeg_decompress_struct) returns different values in different files. In jpgupp.cpp it evaluates to 448 and in jdapimin.c it results to 488. The sanity checks in libjpeg catch this and throw out an error, which is quite reasonable.

I was so far not successful in finding why it happens. I suspected difference between code compiled as c++ and that compiled as plain c. Also, I checked that the jpeglib.h where the struct is defined is preprocessed the same way in both situations. Even when trying with various combinations of parameters (especially -Os/-O3), I was not able to reproduce it in a simpler testcase...

So my quest to hunt this bug down will have to continue. I just wanted to share my above findings in case someone has some idea what could be thee cause and also because it is nice example of what can go wrong when mixing C++ and C

Honza
