Adding Multilib Capability to Slackware on x86_64 Architecture

This article contains instructions on how to create a *true multilib* Slackware64. A multilib 64bit Linux system is capable of running 64bit as well as 32bit software. The Filesystem Hierarchy Standard documents the optimal method to achieve a clean separation between 64bit and 32bit software on a single system. When starting with the development of "Slackware64" (the official port to the x86_64 architecture) we chose to adopt this standard. Therefore Slackware64 has been configured to look for 64bit libraries in `/lib64` and `/usr/lib64` directories. This is why I call Slackware64 "multilib-ready" - even though 32bit libraries will be looked for in `/lib` and `/usr/lib`, Slackware64 does not ship with any 32bit software. There is one more step that must be taken (by you, the user) before Slackware64 can be called "multilib-enabled".

This is accomplished as follows:

1. First we need to switch to multilib versions of
   - *glibc* (i.e. a glibc that supports *running* both 32bit and 64bit binaries), and
   - *gcc* (i.e. able to *compile* 32bit binaries as well as 64bit binaries).
2. Then, system libraries are taken from 32bit Slackware and installed in the 64bit Slackware system alongside their 64bit versions which completes the process of creating a 32bit compatibility software layer.

When Slackware64 was released, it had an advantage over the 64bit "forks" that existed out there. These forks added the 32bit compatibility layer by recompiling a lot of their packages as 32bit binaries. Slackware on the other hand, is a distribution that consists of a 32bit and 64bit release, both of which are being developed in parallel. This means, that you do not have to compile 32-bit packages from scratch in order to add multilib capability to the 64bit system. You simply take them from the 32-bit Slackware package tree!
This was one of the reasons for not adding full multilib to Slackware64 - we create the right preconditions but require the user to act if she needs multilib.
In a section further down, I will explain how you can take a 32-bit Slackware package (say, the "mesa" package) and re-package its content into a "mesa-compat32" package that you can install straight away on Slackware64.

Slackware for the x86_64 architecture (or "*Slackware64*" for short) is a pure 64-bit Operating System, but easily upgradable to multilib. *Out of the box, Slackware64 is only capable of compiling and running 64bit binaries.*

Advantage of a multilib system

I will give some examples of programs that require multilib support on a 64bit Slackware because they will not start or compile on Slackware64 without the 32bit compatibility layer:

- Wine
  Most Windows programs are still 32bit, and in order to run those on Linux with Wine, you need a 32bit version of Wine.
- VirtualBox
  The popular virtual machine software. Although this is (partly) open source it still needs 32-bit compatibility libraries on 64-bit Slackware.

- Steam
  The highly popular gaming platform still needs a 32bit client. Most of the available games are 32bit as well.
- Skype, Citrix client, ...
  These programs are proprietary and closed-source. We have to depend on the developer to make 64bit binaries available. So far, that has not happened for these example programs.

Luckily, 64bit support is becoming more and more common. Adobe was a sore point for a long time, but they eventually released their Flash browser plugin in a 64bit version. Sun (now absorbed by Oracle) revealed a 64bit version of their Java browser plugin. These two events were major triggers to start working on Slackware64.

Obtaining multilib packages

You can download a set of multilib-enabled packages and scripts from my web site: http://slackware.com/~alien/multilib/ .

Apart from several README files (this Wiki article basically is an enhanced version of one of these READMEs), you will find one subdirectory for every 64-bit Slackware release below the toplevel directory "*multilib*". There is another directory called "source". The "*source*" directory contains package sources and SlackBuild scripts.
The stuff that you are really interested in - the binary packages - is available in the *<slackware_release_number>* directory below the toplevel directory. Every such directory also contains a "*slackware64-compat32*" subdirectory where you will find an essential set of converted 32-bit Slackware packages, ready for installing on your 64-bit Slackware.

Keeping your multilib up-to-date

In order to keep up to date, I advise you to keep an eye on the ChangeLog (RSS feed) which I maintain for my multilib packages. Usually, I will have *updated glibc and gcc* packages available within a day after Slackware has updates to gcc and glibc.

Automation:

1. Check out compat32pkg by Sébastien Ballet which automates this process, similar to slackpkg.
2. If you prefer slackpkg for package management, then it is worthwhile to check out slackpkg+, an extension to slackpkg which manages the packages you installed from 3rd-party repositories - including multilib. When properly configured, keeping your multilib is as easy as running:

   ```
   # slackpkg update
   # slackpkg upgrade multilib
   # slackpkg install multilib
   ```

   That last command will show you if any new packages were added to the collection of "compat32" packages, such as llvm-compat32 and orc-compat32 recently.

   - This is how a typical configuration would look like - for a computer running Slackware-current and using Alien BOB's KDE testing repository. The PKGS_PRIORITY ensures that

multilib packages of gcc and glibc have precedence over the Slackware originals. The keyword “multilib” which defines the name for the repository must be the same keyword used in the “slackpkg” commands above. The choice of the word “multilib” is arbitrary, it could just as well have been “compat32”, as long as you use it consistently.
Contents of an example file “/etc/slackpkg/slackpkgplus.conf” would be as follows:

```
SLACKPKGPLUS=on
VERBOSE=1
ALLOW32BIT=off
USEBL=1
WGETOPTS="--timeout=5 --tries=1"
GREYLIST=on
PKGS_PRIORITY=( multilib restricted alienbob ktown )
REPOPLUS=( slackpkgplus multilib restricted alienbob ktown )
MIRRORPLUS['multilib']=http://bear.alienbase.nl/mirrors/people/ali
en/multilib/current/
MIRRORPLUS['alienbob']=http://bear.alienbase.nl/mirrors/people/ali
en/sbrepos/current/x86_64/
MIRRORPLUS['restricted']=http://bear.alienbase.nl/mirrors/people/a
lien/restricted_sbrepos/current/x86_64/
MIRRORPLUS['ktown']=http://bear.alienbase.nl/mirrors/alien-kde/cur
rent/latest/x86_64/
MIRRORPLUS['slackpkgplus']=http://slakfinder.org/slackpkg+/
```

Enabling multilib support on Slackware64

The quick 'n' dirty instructions

This section contains the essential instructions to add full multilib capability to your Slackware64 system. If you want to understand the process in more detail, or need information on how to compile 32bit software in Slackware64, you should also read the sections that follow.
Note that the “#” in front of the commands depicts a *root prompt*.

- Download the packages from my web site (I gave you the URL in the previous section, but this example is using a mirror URL). Suppose you are running Slackware 14.2. You execute:

```
# SLACKVER=14.2
# mkdir multilib
# cd multilib
# lftp -c "open http://bear.alienbase.nl/mirrors/people/alien/multilib/
; mirror -c -e ${SLACKVER}"
# cd ${SLACKVER}
```

- Upgrade your 64bit Slackware “*gcc*” and “*glibc*” packages to my multilib versions.
Run the command

```
# upgradepkg --reinstall --install-new *.t?z
```

after you changed to the directory where you downloaded these packages.
This command will also install an additional package called "*compat32-tools*".

- If you also downloaded a directory called *slackware64-compat32* (my example "`lftp`" command will have done that) then you are lucky, because I did the 32bit package conversion for you already! All you do is run the command:

```
# upgradepkg --install-new slackware64-compat32/*-compat32/*.t?z
```

  which will install all the converted 32-bit Slackware packages (or upgrade them if you already had installed older multilib packages, for instance when you are upgrading to a newer Slackware). That's all!

- If you can not find a subdirectory called *slackware64-compat32* then either you did not download it, or the download mirror did not provide it. In this case, you have to do the 32-bit package conversion yourself. Not difficult at all, it takes a few minutes:
  - The fastest is if you have a local directory with original 32-bit Slackware packages available (also called a *local mirror*). Those who bought an official Slackware DVD can simply use that DVD: it is dual-sided and 32bit Slackware is on one of the sides. For the sake of this example I will assume that you have a local 32bit Slackware directory tree available at "`/home/ftp/pub/slackware/slackware-14.2/slackware/`".
    There should be sub-directories called 'a', 'ap', 'd', 'l', 'n', 'x' immediately below this directory. (If you have mounted a Slackware DVD, your directory will probably be "`/media/SlackDVD/slackware/`" but I will not use that in the example commands below).
  - Create a new empty directory (let us call it 'slackware64-compat32') and change into it:

```
# mkdir slackware64-compat32 ; cd slackware64-compat32
```

  - Run the following command to create a set of 32bit compatibility packages, using the directory to the official 32bit Slackware packages as input:

```
# massconvert32.sh -i
/home/ftp/pub/slackware/slackware-14.2/slackware/
```

  - The previous step takes a while. When it ends, proceed to install the 90 MB of freshly converted 32-bit Slackware packages which were created in subdirectories below your *current directory*:

```
# upgradepkg --install-new *-compat32/*.t?z
```

  - Done! You can now start downloading, installing and running 32bit programs. This was not so hard, was it?

If you use a package manager like *slackpkg* you will have to add all the `glibc` and `gcc` package names to its package blacklist. If you do not take this precaution, you run the risk of your package manager accidentally replacing your multilib versions with Slackware's original pure 64-bit versions! If you run Slackware 13.37 or newer, then slackpkg supports regular expressions in the blacklist file. In that case, a single line in `/etc/slackpkg/blacklist` will be enough to blacklist all of my packages (including multilib gcc and glibc packages and all `compat32` packages):

```
[0-9]+alien
```

```
[0-9]+compat32
```

On the other hand, if you are using the slackpkg extension called slackpkg+ then you should definitely **not** blacklist these packages, because that prevents slackpkg+ from managing them!

If you are running Slackware 13.1 or newer, and downloaded the compat32-tools package for that release, the *massconvert32.sh* script can use a remote webserver to download the 32-bit Slackware packages from, instead of requiring a local Slackware mirror or a DVD. You use the "-u" parameter to specify the remote URL like this:

```
# massconvert32.sh -u http://someserver.org/path/to/slackware-14.2/slackware
```

Detailed instructions

Upgrading glibc and gcc

The following glibc/gcc packages are replacements for - *not additions to* - standard Slackware packages. You use the "`upgradepkg`" program to upgrade to my multilib versions of gcc and glibc. You will need these in order to run (glibc), and build (gcc), 32-bit software on your 64-bit Slackware computer:

Slackware64 13.0

- The gcc compiler suite:
  - gcc-4.3.3_multilib-x86_64-4alien.txz
  - gcc-g++-4.3.3_multilib-x86_64-4alien.txz
  - gcc-gfortran-4.3.3_multilib-x86_64-4alien.txz
  - gcc-gnat-4.3.3_multilib-x86_64-4alien.txz
  - gcc-java-4.3.3_multilib-x86_64-4alien.txz
  - gcc-objc-4.3.3_multilib-x86_64-4alien.txz
- The GNU libc libraries:
  - glibc-2.9_multilib-x86_64-3alien.txz
  - glibc-i18n-2.9_multilib-x86_64-3alien.txz
  - glibc-profile-2.9_multilib-x86_64-3alien.txz
  - glibc-solibs-2.9_multilib-x86_64-3alien.txz
  - glibc-zoneinfo-2.9_multilib-noarch-3alien.txz

Slackware64 13.1

- The gcc compiler suite:
  - gcc-4.4.4_multilib-x86_64-1alien.txz
  - gcc-g++-4.4.4_multilib-x86_64-1alien.txz
  - gcc-gfortran-4.4.4_multilib-x86_64-1alien.txz
  - gcc-gnat-4.4.4_multilib-x86_64-1alien.txz
  - gcc-java-4.4.4_multilib-x86_64-1alien.txz
  - gcc-objc-4.4.4_multilib-x86_64-1alien.txz
- The GNU libc libraries:

    - glibc-2.11.1_multilib-x86_64-3alien.txz
    - glibc-i18n-2.11.1_multilib-x86_64-3alien.txz
    - glibc-profile-2.11.1_multilib-x86_64-3alien.txz
    - glibc-solibs-2.11.1_multilib-x86_64-3alien.txz
    - glibc-zoneinfo-2.11.1_multilib-noarch-3alien.txz

Slackware64 13.37

- The gcc compiler suite:
    - gcc-4.5.2_multilib-x86_64-2alien.txz
    - gcc-g++-4.5.2_multilib-x86_64-2alien.txz
    - gcc-gfortran-4.5.2_multilib-x86_64-2alien.txz
    - gcc-gnat-4.5.2_multilib-x86_64-2alien.txz
    - gcc-java-4.5.2_multilib-x86_64-2alien.txz
    - gcc-objc-4.5.2_multilib-x86_64-2alien.txz
- The GNU libc libraries:
    - glibc-2.13_multilib-x86_64-7alien.txz
    - glibc-i18n-2.13_multilib-x86_64-7alien.txz
    - glibc-profile-2.13_multilib-x86_64-7alien.txz
    - glibc-solibs-2.13_multilib-x86_64-7alien.txz

Slackware64 14.0

- The gcc compiler suite:
    - gcc-g++-4.7.1_multilib-x86_64-1alien.txz
    - gcc-gfortran-4.7.1_multilib-x86_64-1alien.txz
    - gcc-gnat-4.7.1_multilib-x86_64-1alien.txz
    - gcc-go-4.7.1_multilib-x86_64-1alien.txz
    - gcc-java-4.7.1_multilib-x86_64-1alien.txz
    - gcc-objc-4.7.1_multilib-x86_64-1alien.txz
- The GNU libc libraries:
    - glibc-2.15_multilib-x86_64-9alien.txz
    - glibc-i18n-2.15_multilib-x86_64-9alien.txz
    - glibc-profile-2.15_multilib-x86_64-9alien.txz
    - glibc-solibs-2.15_multilib-x86_64-9alien.txz

Slackware64 14.1

- The gcc compiler suite:
    - gcc-4.8.2_multilib-x86_64-1alien.txz
    - gcc-g++-4.8.2_multilib-x86_64-1alien.txz
    - gcc-gfortran-4.8.2_multilib-x86_64-1alien.txz
    - gcc-gnat-4.8.2_multilib-x86_64-1alien.txz
    - gcc-go-4.8.2_multilib-x86_64-1alien.txz
    - gcc-java-4.8.2_multilib-x86_64-1alien.txz
    - gcc-objc-4.8.2_multilib-x86_64-1alien.txz
- The GNU libc libraries:
    - glibc-2.17_multilib-x86_64-10alien.txz

    - glibc-i18n-2.17_multilib-x86_64-10alien.txz
    - glibc-profile-2.17_multilib-x86_64-10alien.txz
    - glibc-solibs-2.17_multilib-x86_64-10alien.txz

Slackware64 14.2

- The gcc compiler suite:
    - gcc-5.3.0_multilib-x86_64-3alien.txz
    - gcc-g++-5.3.0_multilib-x86_64-3alien.txz
    - gcc-gfortran-5.3.0_multilib-x86_64-3alien.txz
    - gcc-gnat-5.3.0_multilib-x86_64-3alien.txz
    - gcc-go-5.3.0_multilib-x86_64-3alien.txz
    - gcc-java-5.3.0_multilib-x86_64-3alien.txz
    - gcc-objc-5.3.0_multilib-x86_64-3alien.txz
- The GNU libc libraries:
    - glibc-2.23_multilib-x86_64-2alien.txz
    - glibc-i18n-2.23_multilib-x86_64-2alien.txz
    - glibc-profile-2.23_multilib-x86_64-2alien.txz
    - glibc-solibs-2.23_multilib-x86_64-2alien.txz

Slackware64 current

- As long as you don't see a separate directory named "*current*" you can just use the files in the directory for the most recent stable release.

- The gcc compiler suite:
    - gcc-7.1.0_multilib-x86_64-2alien.txz
    - gcc-brig-7.1.0_multilib-x86_64-2alien.txz
    - gcc-g++-7.1.0_multilib-x86_64-2alien.txz
    - gcc-gfortran-7.1.0_multilib-x86_64-2alien.txz
    - gcc-gnat-7.1.0_multilib-x86_64-2alien.txz
    - gcc-go-7.1.0_multilib-x86_64-2alien.txz
    - gcc-objc-7.1.0_multilib-x86_64-2alien.txz
- The GNU libc libraries:
    - glibc-2.25_multilib-x86_64-3alien.txz
    - glibc-i18n-2.25_multilib-x86_64-3alien.txz
    - glibc-profile-2.25_multilib-x86_64-3alien.txz
    - glibc-solibs-2.25_multilib-x86_64-3alien.txz

Since the update to gcc 7, there is no more `gcc-java` package because its development has ceased. The `glibc-zoneinfo` package is not a part of multilib, since it does not contain code. You need to install Slackware's stock `glibc-zoneinfo` package.

All releases of Slackware

There is one additional package that you need to install using the "installpkg" program. The actual version may vary for each release of Slackware, but the package can be found in the same directory where you also find the multilib versions of `gcc` and `glibc`:

- The "32bit toolkit" (scripts that facilitate the creation of 32bit packages)

- compat32-tools-3.7-noarch-1alien.tgz

Adding 32-bit Slackware libraries

The upgrade of glibc and gcc which I described in the previous section changes your system from "*multilib-ready*" to "*multilib-enabled*".
Now, all you need to do is to install 32bit versions of Slackware's system software so that future 32bit programs that you are going to install and/or compile will find all the 32bit libraries they need in order to work.

This is not as simple as grabbing 32bit Slackware packages and installing them in Slackware64:

- In the first place, you will end up with multiple packages carrying the same name (two 'mesa' packages, two 'zlib' packages, etc…) which will be confusing to you as well as to the *slackpkg* package manager.
- And furthermore, if the 32bit package contains binaries (something like `/usr/bin/foo`), they will overwrite their 64bit counterparts when you install the 32bit package on top. It will seriously mess up your system if that happens.

A little bit of extra care is required so that unnecessary/unwanted files are stripped from the 32bit packages before you install them. What you need, is a 32bit package that does not conflict with whatever is already present in 64bit Slackware. Hence the name "32bit compatibility package".

I decided that it would be a waste of download bandwidth if I created 32bit compatibility versions of Slackware packages myself. After all, you have probably bought the Slackware 14.2 DVD so you already possess both 64bit and 32bit versions of Slackware… or else the 32bit Slackware tree is available for free download of course 

Instead, I wrote a few scripts (parts of the script code were written by Fred Emmott of Slamd64 fame) and wrapped these into a "*compat32-tools*" package. Their purpose is to let you extract the content from any 32bit Slackware package and use that to create a new package which you can safely install on your 64bit Slackware.

This "*compat32-tools*" package needs some explanation.
Please read the detailed '*README*' file in the `/usr/doc/compat32-tools-*/` directory, it will help you on your way. These are the three useful scripts which the package installs:

- */etc/profile.d/32dev.sh*
  This is the same script that comes with Slamd64. It reconfigures your shell environment so that it will be easier for you to compile 32-bit software (by preferring the 32-bit compilers and libraries over their 64-bit versions)

- *convertpkg-compat32*
  This script takes a 32-bit Slackware package and converts it to a '-compat32' package that you can safely install (using "installpkg") on Slackware64, alongside the 64-bit version of the same software package. For instance: suppose you need 32bit libraries that are in the mesa package. You take the mesa package from 32-bit Slackware (`x/mesa-7.5-i486-1.txz`) and then run

  ```
  # convertpkg-compat32 -i /path/to/mesa-7.5-i486-1.txz
  ```

  which will create a new package called `mesa-compat32-7.5-x86_64-1compat32.txz`. This

new package (which is created in your `/tmp` directory unless you specified another destination) is basically the old 32bit package, but stripped from non-essential stuff. The changed basename (*mesa* becomes *mesa-compat32*) allows you to install this new package in Slackware64 where it will co-exist with the 64bit *mesa* package, not overwriting any files.
The script leaves temporary files in the directory "`/tmp/package-<prgnam>-compat32`" which you can safely delete.

- *massconvert32.sh*
  This script contains an internal list of what I consider the essential subset of 32bit Slackware packages. It uses the above "*convertpkg-compat32*" script to grab every package that is on this internal list, and convert these into '-compat32' packages.
  You need to run this script only once, for example like this (the example assumes that you mounted your 32bit Slackware DVD on `/mnt/dvd`):

```
# massconvert32.sh -i /mnt/dvd/slackware -d ~/compat32
```

  This action will result in about 150 MB of new packages which you will find inside the newly created directory `~/compat32` (the directory's name is arbitrary of course, I chose it for the sake of this example). These packages comprise the 32bit component of your multilib Slackware64 system.
  They should be installed using "`installpkg`", and they give you a pretty complete 32-bit compatibility layer on top of Slackware64:

```
# installpkg ~/compat32/*/*.t?z
```

  If you are upgrading from an earlier version of these packages (because for instance you upgraded your 64-bit Slackware to a newer release) then you do not use "`installpkg`" of course, but "`upgradepkg –install-new`" instead:

```
# upgradepkg --install-new ~/compat32/*/*.t?z
```

  The "*–install-new*" parameter is needed to install the new *compat32* packages which were added between releases.

When installing the *compat32* packages you will notice that some will show errors about missing files in `/etc`. This is "by design", and these errors can be ignored. These messages are caused by the fact that files in `/etc` are removed from a "-compat32" package during conversion (except for *pango* and *gtk+2*). I assume that files in `/etc` will already have been installed by the original 64bit packages. An example of these "errors" for the `cups-compat32` package:

```
Executing install script for cups-compat32-1.3.11-x86_64-1.txz.
install/doinst.sh: line 5: [: too many arguments
cat: etc/cups/interfaces: Is a directory
cat: etc/cups/ppd: Is a directory
cat: etc/cups/ssl: Is a directory
cat: etc/cups/*.new: No such file or directory
cat: etc/dbus-1/system.d/cups.conf.new: No such file or directory
chmod: cannot access `etc/rc.d/rc.cups.new': No such file or directory
cat: etc/rc.d/rc.cups.new: No such file or directory
```

```
Package cups-compat32-1.3.11-x86_64-1.txz installed.
```

If you were considering to use the `convertpkg-compat32` script to convert a **non-Slackware** package to a *-compat32* package, I must strongly advise against this. The script is written with a single purpose and that is to make 32bit versions of the official Slackware64 binaries/libraries available in a multilib setup. As such, the script will remove a lot of stuff that is present in the original 32bit package - stuff which is expected to have been installed as part of the 64bit version of the package.
In almost all cases where you have downloaded a non-Slackware 32bit package and want to make it work on Slackware64, the best way is to find the sources and build a 64bit version of the package. Alternatively, just *install the original* 32bit package instead of trying to “convert it” and then run it from the commandline to find out any missing 32bit libraries you may still have to extract from an official Slackware package.

Running 32-bit programs

Running a pre-compiled 32-bit program is easy after you've done the above system preparation. Just download, install and run it!

At times, you may run into a program that requires a certain 32-bit Slackware library that you do not yet have available. In that case, find out which 32bit Slackware package contains this missing library. Use the “*convertpkg-compat32*” script to convert that original 32bit Slackware package and install the resulting 32bit “*compatibility*” package on Slackware64.

Compiling 32-bit programs

In case you need to compile a 32-bit program (wine and grub are two examples of open source programs that are 32-bit only) you first configure root's shell environment by running the command:

```
# . /etc/profile.d/32dev.sh
```

Note the 'dot' at the beginning of the line - that is actually part of the commandline! The use of the dot is equivalent to the 'source' command.
Running this command changes or creates several environment variables. The effect of this is, that 32-bit versions of binaries are preferred over 64bit binaries when you compile source code - you will be running a 32bit compilation. The effect will last until you logout from your root shell.

In this changed environment, you will be able to use standard SlackBuilds to build 32-bit packages for Slackware64. There's a couple of things to keep in mind:

1. You have to define the ARCH variable as 'i486' because even on your 'x86_64' computer you are compiling a 32-bit program!
   This is related to the *triplet* of “$ARCH-slackware-linux” which is normally used in the “configure” command.
    1. As an exception, you will have to compile the “wine” package with 'ARCH=x86_64' because you will install this package directly on your multilib computer without converting to a 'compat32' package.
2. If you want to install this 32-bit package on Slackware64-multilib you will have to convert it to a 'compat32' package:

```
# convertpkg-compat32 -i /path/to/your/fresh/foo-VERSION-i486-BUILD.tgz
# upgradepkg --install-new /tmp/foo-compat32-VERSION-x86_64-
BUILDcompat32.txz
```

Caveats

- After installing the "`-compat32`" packages, you may have to re-install your binary *Nvidia* or *Ati* video X.Org drivers. These driver packages contain both 64bit and 32bit libraries to be maximally useful on a 64bit multilib OS. If you installed the driver files for both architectures, the "`mesa-compat32`" package will overwrite some of the 32bit library files.

  On the other hand, if you originally *only* installed the 64bit driver libraries for your Nvidia/Ati card, it is recommended after installation of the *multilib* packages, to re-install the binary driver package. This time, choose to install the 32bit driver files as well.

  The graphical 32bit applications that you are going to run on your multilib installation will require these 32bit driver libraries. Crashes are likely to occur if you fail to install the correct files.

- If you want to compile your 64bit kernel yourself, be sure to compile 32bit emulation capability into it or else multilib will mysteriously fail. You will need this piece of kernel configuration: **CONFIG_IA32_EMULATION**

Packages converted by massconvert32.sh

This is the list of packages that is converted into "*-compat32*" versions by the `massconvert32.sh` script. Note that some of these packages are not part of Slackware 13.0 or 13.1, they were added in a later Slackware version so they will produce a "**** FAIL: package '*package_name*' was not found!" message when you run the script on an older release. The other way round is true as well - some packages have been removed in later versions of Slackware and they will also trigger the "***** FAIL: package '*package_name*' was not found!"* message. Don't worry about that.

```
# The A/ series:

aaa_elflibs
attr
bzip2
cups
cxxlibs
dbus
e2fsprogs
eudev
libgudev
openssl-solibs
udev
util-linux
xz
```

```
# The AP/ series:

cups
cups-filters
flac
mariadb
mpg123
mysql
sqlite

# The D/ series:

libtool
llvm
opencl-headers

# The L/ series:

SDL2
alsa-lib
alsa-oss
alsa-plugins
atk
audiofile
cairo
dbus-glib
elfutils
esound
expat
ffmpeg
fftw
freetype
fribidi
gamin
gc
gdk-pixbuf2
giflib
glib2
gmp
gnome-keyring
gtk+2
gst-plugins-base
gst-plugins-base0
gst-plugins-good
gst-plugins-good0
gst-plugins-libav
gstreamer
gstreamer0
hal
harfbuzz
```

```
icu4c
jasper
json-c
lame
lcms
lcms2
libaio
libart_lgpl
libasyncns
libclc
libedit
libelf
libexif
libffi
libglade
libgphoto2
libidn
libieee1284
libjpeg
libjpeg-turbo
libmng
libmpc
libnl3
libnotify
libogg
libpcap
libpng
libsamplerate
libsndfile
libtasn1
libtermcap
libtiff
libunistring
libusb
libvorbis
libxml2
libxslt
lzo
ncurses
ocl-icd
openjpeg
orc
pango
popt
pulseaudio
python-six
qt
readline
sbc
sdl
seamonkey-solibs
```

```
speexdsp
startup-notification
svgalib
v4l-utils
zlib

# The N/ series:

curl
cyrus-sasl
gnutls
libgcrypt
libgpg-error
libtirpc
nettle
openldap-client
openssl
p11-kit
samba

# The X/ series:

fontconfig
freeglut
glew
glu
libFS
libICE
libSM
libX11
libXScrnSaver
libXTrap
libXau
libXaw
libXcomposite
libXcursor
libXdamage
libXdmcp
libXevie
libXext
libXfixes
libXfont
libXfont2
libXfontcache
libXft
libXi
libXinerama
libXmu
libXp
libXpm
libXprintUtil
```

```
libXrandr
libXrender
libXres
libXt
libXtst
libXv
libXvMC
libXxf86dga
libXxf86misc
libXxf86vm
libdmx
libdrm
libepoxy
libfontenc
libinput
libpciaccess
libva
libva-intel-driver
libvdpau
libxcb
libxshmfence
mesa
pixman
vulkan-sdk
xcb-util

# The XAP/ series:

sane
```

Multilib download mirrors

You can download the multilib packages from (at least) these locations:

- http://slackware.com/~alien/multilib/
- http://bear.alienbase.nl/mirrors/people/alien/multilib/
- http://slackware.uk/people/alien/multilib/
- http://alien.slackbook.org/slackware/multilib/
- http://slackbuilds.org/mirror/alien/multilib/

3rd party support tools

- Sébastien Ballet has written a tool called *compat32pkg*. On his web site he has *compat32pkg* available for download as well as extensive documentation about how to use it on Slackware64. I will quote the site:
  "Compat32pkg is an automated tool that provides all the necessary for managing (converting, installing, upgrading, removing) the 32-bit part of AlienBob's multilib for slackware-64, and all 32-bit packages from Slackware-32 for which users could find a needs into a 64-bit

environment, like firefox, seamonkey, jre,...”

- There is also slackpkg+, written by Matteo Rossini (nicknamed zerouno) with contributions from (among others) Sébastien Ballet. This is a plugin for Slackware's own slackpkg which adds the capability for installing packages from external (3rd-party) unofficial Slackware repositories. It has good support for adding multilib to your 64-bit Slackware and keeping it up to date.

Translations

- Bruno Russo translated this article to portuguese (brazil): http://www.brunorusso.eti.br/slackware/doku.php?id=multilib_para_o_slackware_x86_64
- Mehdi Esmaeelpour translated this article to persian: http://www.slack-world.com/index.php/articles/43-general-system/85-multilib-slackware64
- Patrick FONIO and Sebastien BALLET translated this article to french: http://wiki.slackware-fr.org/avance:articles:slackware64-multilib

Acknowledgements

- A lot of thanks should go to Fred Emmott, who created Slamd64, the original unofficial 64-bit fork of Slackware. Although Slackware64 was not based on Fred's work, I still learnt most of what I know about setting up the 32-bit part of a multilib Linux from his writings that are found in Slamd64.
  Note that Slamd64 had separate 64bit and 32bit gcc/glibc multilib packages. However, I believe that it is cleaner to keep these essential multilib packages undivided. I followed the concept already used in Slackware64's own *binutils* package, which has 64-bit and 32-bit multilib capability bundled into one package.
- Cross Linux From Scratch.
  The CLFS Wiki (http://trac.cross-lfs.org/wiki/read#ReadtheCrossLinuxFromScratchBookOnline) is a 'must-read' if you want to understand how to port Linux to a new architecture. I took several ideas, concepts and patches from them when creating Slackware64 from scratch, and again when I created my multilib gcc/glibc packages from scratch (my README on this multilib-from-scratch is available in the ./source directory).

Have fun!

Eric

Sources

- The original article, written by Eric Hameleers, is at http://alien.slackbook.org/dokuwiki/doku.php?id=slackware:multilib

slackware, multilib, author alienbob