Fakeroot in Scratchbox

Timo Savola

tsavola@movial.fi

Fakeroot in Scratchbox

by Timo Savola

Revision history

Version:	Author:	Description:
2005-02-06	Savola	Based on *Device tools*

Table of Contents

1. Introduction to fakeroot **1**
2. Fakeroot in Scratchbox **2**
2.1. Known issues 2
3. Debugging **4**
4. Building fakeroot **5**
5. Implementation of network fakeroot **7**
References **8**
A. fakeroot 1.2.3 manual page **9**

Chapter 1. Introduction to fakeroot

Fakeroot [1] is a utility that runs programs in an environment that looks as if they were run with super-user privileges. It is used primarily for setting file ownerships and modes before packaging them. You can for example create device nodes and store them in a tarball while logged in as a normal user. Of course, the programs run from a fakeroot session cannot really do privileged system calls; fakeroot keeps an in-memory database of file ownerships and such things.

Fakeroot was developed by the Debian Project [2] to help in building Debian packages. The Debian packaging system needs a root environment so that it would be as easy as possible to set up ownerships and permissions.

Fakeroot is released under the GNU General Public License version 2 [3].

Chapter 2. Fakeroot in Scratchbox

Scratchbox [4] introduces new requirements for fakeroot. During the development of Scratchbox an enhanced version of fakeroot was developed with the name *fakeroot-net*. It was later merged with the upstream project and nowadays Scratchbox uses the upstream codebase.

Differences between the default and the Scratchbox version are:

- When using sbrsh [5] to implement CPU-transparency in Scratchbox, the command execution can jump from host to target within a fakeroot session. Since both ends use the same filesystems (via NFS), they must also use the same fakeroot session. This is not possible with the original design that uses SYSV IPC. The Scratchbox version uses TCP/IP sockets for its internal communication. (The TCP version of the fakeroot command is also available in the Debian package with the name **fakeroot-tcp**.)

  Using TCP sockets in fakeroot is not enough to implement network-transparent fakeroot sessions. The sbrsh server (sbrshd) is used to filter the information passed between the remote fakeroot environment and the fakeroot daemon (faked) that keeps the database. The reason for this is explained in Chapter 5.

- Fakeroot supports saving and loading its internal database in a file. The default file format uses inode numbers to identify files. Scratchbox uses full path names instead of inodes so that it can reliably check the existence of the files when loading a database. (The Debian binary package does not ship this version but the functionality is included in the source package.)

Scratchbox provides the fakeroot command, the fakeroot daemon (faked) and a host version of the fakeroot library (libfakeroot). They are sufficient for running host tools in fakeroot, but a target version of libfakeroot needs to be installed for each Scratchbox target in order to run target binaries in fakeroot. *Installing Scratchbox* [6] describes how to do that. Scratchbox's fakeroot is compatible with the libfakeroot provided by the Debian package, so you can use that aswell.

> **Note:** As described above, the fakeroot daemon provided by the Debian package does not use the same database format as Scratchbox's version. This should not be a problem though, since thanks to Scratchbox's binary redirection feature the host version of the fakeroot command is normally used even when the target version is installed.

Refer to fakeroot's manual page (Appendix A) for usage instructions.

2.1. Known issues

The fakeroot environment is imposed upon a process by using the C-library's LD_PRELOAD environment variable. libfakeroot is preloaded by the dynamic linker whenever it loads a binary. This

means that fakeroot does not work with statically linked binaries.

There is also another side-effect. Since libfakeroot is loaded into the same process image with the “victim” program, they share the same file descriptor table. Some programs (such as the configure scripts) use hard-coded descriptor numbers. libfakeroot needs one file descriptor for its communication socket, and if the program starts to use the same file descriptor, there will be trouble. fakeroot tries to monitor the status of its descriptor so that it can open a new socket if the descriptor has been changed. If you start seeing messages about hijacked file descriptors, you can try to make fakeroot use some other file descriptor with the **--fd-base** option. Its default value is (*descriptor_table_size* - 100).

Chapter 3. Debugging

The fakeroot daemon can be launched with debug enabled and left running on the foreground:

```
$ faked --debug --foreground
33366:5027
```

The first number is the TCP/IP port it listens to, and the second number is its process ID. Now, in another terminal, setup a fakeroot session manually that uses the daemon we started:

```
$ export FAKEROOTKEY=33366
$ export LD_PRELOAD=/scratchbox/tools/lib/libfakeroot-tcp.so.0
```

Now you can run programs in the hand-made fakeroot session and see the daemon's cryptic debug output in the other terminal. This way you can also use a debugger to debug a program within a fakeroot environment.

> **Note:** `/scratchbox/tools/lib/libfakeroot-tcp.so.0` is the host version. If you are running target binaries, you should set LD_PRELOAD to `/usr/lib/libfakeroot/libfakeroot-tcp.so.0`.

When using a remote fakeroot session, the communication can be traced using the sbrsh daemon's debug log. See *Scratchbox Remote Shell* [5] for instructions.

Chapter 4. Building fakeroot

This chapter contains instructions for building fakeroot from source code using Scratchbox's configuration options. You shouldn't normally need to do that, since fakeroot is included in Scratchbox and all toolchains ship libfakeroot binaries for their target architectures. See *Installing Scratchbox* [6] and *Scratchbox toolchains* [7] for more information.

Fakeroot should be cross-compiled inside Scratchbox. The fakeroot source package is available in the `/scratchbox/packages` directory in the Scratchbox installation, but you can also download it from Debian [7].

Here fakeroot is compiled for a preconfigured Scratchbox target:

1. Extract the fakeroot source package:

   ```
   [sbox-HOST: ~] > tar xfz /scratchbox/packages/fakeroot_1.2.3.tar.gz
   ```

2. Select the target your wish to compile for:

   ```
   [sbox-HOST: ~] > sb-conf select ARM
   ```

3. Go to the source directory:

   ```
   [sbox-ARM: ~] > cd fakeroot-1.2.3
   ```

4. Configure fakeroot using the options used by Scratchbox:

   ```
   [sbox-ARM: ~/fakeroot-1.2.3] > ./configure \
   --prefix=/usr --mandir=/usr/share/man --libdir=/usr/lib/libfakeroot \
   --program-suffix=-tcp --with-ipc=tcp --with-dbformat=path
   ```

   Note: Fakeroot uses the `/usr/lib/libfakeroot` directory for its *real* libraries. A *fake* library is installed to `/usr/lib` to work around a bug in an old version of the dynamic linker.

5. Build the *real* libfakeroot-tcp and install it along with the fakeroot-tcp command on the target:

   ```
   [sbox-ARM: ~/fakeroot-1.2.3] > make
   [sbox-ARM: ~/fakeroot-1.2.3] > make install
   ```

6. Clean the configuration and go to the fake directory:

   ```
   [sbox-ARM: ~/fakeroot-1.2.3] > make distclean
   [sbox-ARM: ~/fakeroot-1.2.3] > cd fake
   ```

7. Configure the *fake* fakeroot:

```
[sbox-ARM: ~/fakeroot-1.2.3/fake] > ../configure \
--prefix=/usr --mandir=/usr/share/man \
--program-suffix=-tcp
```

8. Build and install the *fake* libfakeroot-tcp:

```
[sbox-ARM: ~/fakeroot-1.2.3/fake] > make
[sbox-ARM: ~/fakeroot-1.2.3/fake] > make install
```

9. We won't be building the non-TCP version so let's link it to the TCP version:

```
[sbox-ARM: ~/fakeroot-1.2.3/fake] > ln -sf fakeroot-tcp /usr/bin/fakeroot
```

10. If you are using the Debian devkit, you can also build a binary package for Debian:

```
[sbox-ARM: ~/fakeroot-1.2.3/fake] > make distclean
[sbox-ARM: ~/fakeroot-1.2.3/fake] > cd ..
[sbox-ARM: ~/fakeroot-1.2.3] > dpkg-buildpackage -rfakeroot -b
```

Note: It is important to note that the Debian package uses the non-TCP version as the default fakeroot command. Also, neither version is configured with the `--with-dbformat=path` option. You can change the configure options by editing the `debian/rules` file. If you do that, you should also change the package name and/or the package version to reflect the incompatibility with the standard Debian package.

Note: The Debian package needs the "sharutils" package for running its tests. Scratchbox does not provide this package, so you might first need to install it on the target:

```
[sbox-ARM: ~/fakeroot-1.2.3] > apt-get update
[sbox-ARM: ~/fakeroot-1.2.3] > apt-get install sharutils
```

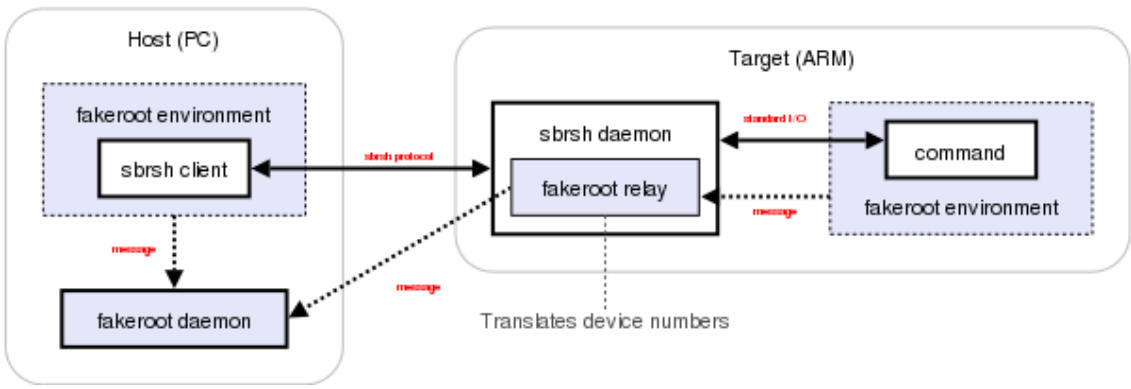
Chapter 5. Implementation of network fakeroot

faked maintains a list of entries based on their device and inode numbers of the files that have been modified during a fakeroot session. The entries contain a data structure that is essentially the same as the one used by the `stat` system call. The TCP version introduces an additional `remote` field in the entry, which works like a “namespace” for the devices and inodes. All files on the local filesystems belog to the default namespace (`remote` is not set).

When a remote command is run within a fakeroot session, sbrsh resolves the device numbers of the NFS filesystems that are listed in its config file for the used target. If they are not exported by the local host but some third host, it tries to find out if the NFS filesystems are mounted on the local host and use the device numbers of the mount points.

sbrshd receives the list of mount entries and finds out what their device numbers are on the target device. Then it creates a *relay* process that listens for connections from local fakeroot sessions. When it receives one, it makes a corresponding connection to the faked running on the Scratchbox host. It maintains as many connection pairs as there are processes running within the local fakeroot session. The relay copies messages from the local session to the remote daemon and responses from the daemon to the session, and translates the device numbers in the messages between the local and remote device number “spaces”.

If the relay finds an unlisted device number in one of the incoming messages, it does not translate it but sets the value of the `remote` field to the IP address of the host it is running at. This way faked can serve unknown filesystems without the danger of device number/inode collisions.



References

[1] *fakeroot (http://fakeroot.alioth.debian.org/)*.

[2] *The Debian Project (http://www.debian.org/)*.

[3] *GNU General Public License (http://www.gnu.org/copyleft/gpl.html)*.

[4] *Scratchbox website (http://scratchbox.org/)*.

[5] *Scratchbox Remote Shell (http://scratchbox.org/documentation/docbook/sbrsh.html)*, Timo Savola.

[6] *Installing Scratchbox (http://scratchbox.org/documentation/docbook/installdoc.html)*, Valtteri Rahkonen.

[7] *Scratchbox toolchains (http://scratchbox.org/documentation/docbook/toolchain.html)*, Ricardo Kekki.

[7] *Debian — fakeroot (http://packages.debian.org/fakeroot)*.

Appendix A. fakeroot 1.2.3 manual page

NAME

fakeroot - run a command in an environment faking root privileges for file manipulation

SYNOPSIS

fakeroot [-l|--lib library] [--faked faked-binary] [-i load-file] [-s save-file] [-u|--unknown-is-real] [-b|--fd-base] [-h|--help] [-v|--version] [--] [command]

DESCRIPTION

fakeroot runs a command in an environment wherein it appears to have root privileges for file manipulation. This is useful for allowing users to create archives (tar, ar, .deb etc.) with files in them with root permissions/ownership. Without fakeroot one would need to have root privileges to create the constituent files of the archives with the correct permissions and ownership, and then pack them up, or one would have to construct the archives directly, without using the archiver.

fakeroot works by replacing the file manipulation library functions (chmod(2), stat(2) etc.) by ones that simulate the effect the real library functions would have had, had the user really been root. These wrapper functions are in a shared library /usr/lib/libfakeroot.so* which is loaded through the LD_PRELOAD mechanism of the dynamic loader. (See ld.so(8))

If you intend to build packages with fakeroot, please try building the fakeroot package first: the "debian/rules build" stage has a few tests (testing mostly for bugs in old fakeroot versions). If those tests fail (for example because you have certain libc5 programs on your system), other packages you build with fakeroot will quite likely fail too, but possibly in much more subtle ways.

Also, note that it's best not to do the building of the binaries themselves under fakeroot. Especially configure and friends don't like it when the system suddenly behaves differently from what they expect. (or, they randomly unset some environment variables, some of which fakeroot needs).

OPTIONS

-l library, --lib library

Specify an alternative wrapper library.

--faked binary

Specify an alternative binary to use as faked.

[--] command
Any command you want to be ran as fakeroot. Use '--' if in the command you have other options that may confuse fakeroot's option parsing.

-s save-file
Save the fakeroot environment to save-file on exit. This file can be used to restore the environment later using -i. However, this file will leak and fakeroot will behave in odd ways unless you leave the files touched inside the fakeroot alone when outside the environment. Still, this can be useful. For example, it can be used with rsync(1) to back up and restore whole directory trees complete with user, group and device information without needing to be root. See /usr/share/doc/fakeroot/README.saving for more details.

-i load-file
Load a fakeroot environment previously saved using -s from load-file. Note that this does not implicitly save the file, use -s as well for that behaviour. Using the same file for both -i and -s in a single fakeroot invocation is safe.

-u, --unknown-is-real
Use the real ownership of files previously unknown to fakeroot instead of pretending they are owned by root:root.

-b fd Specify fd base (TCP mode only). fd is the minimum file descriptor number to use for TCP connections; this may be important to avoid conflicts with the file descriptors used by the programs being run under fakeroot.

-h Display help.

-v Display version.

EXAMPLES
Here is an example session with fakeroot. Notice that inside the fake root environment file manipulation that requires root privileges succeeds, but is not really happening.

```
$  whoami
joost
$ fakeroot /bin/bash
#  whoami
root
# mknod hda3 b 3 1
# ls -ld hda3
brw-r--r--   1 root     root       3,   1 Jul  2 22:58 hda3
# chown joost:root hda3
# ls -ld hda3
```

```
brw-r--r--   1 joost    root       3,   1 Jul  2 22:58 hda3
# ls -ld /
drwxr-xr-x  20 root     root         1024 Jun 17 21:50 /
# chown joost:users /
# chmod a+w /
# ls -ld /
drwxrwxrwx  20 joost    users        1024 Jun 17 21:50 /
# exit
$ ls -ld /
drwxr-xr-x  20 root     root         1024 Jun 17 21:50 //
$ ls -ld hda3
-rw-r--r--   1 joost    users           0 Jul  2 22:58 hda3
```

Only the effects that user joost could do anyway happen for real.

fakeroot was specifically written to enable users to create Debian GNU/Linux packages (in the deb(5) format) without giving them root privileges. This can be done by commands like dpkg-buildpackage -rfakeroot or debuild -rfakeroot (actually, -rfakeroot is default in debuild nowadays, so you don't need that argument).

SECURITY ASPECTS

fakeroot is a regular, non-setuid program. It does not enhance a user's privileges, or decrease the system's security.

FILES

/usr/lib/libfakeroot/libfakeroot.so* The shared library containing the wrapper functions.

ENVIRONMENT

FAKEROOTKEY

The key used to communicate with the fakeroot daemon. Any program started with the right LD_PRELOAD and a FAKEROOTKEY of a running daemon will automatically connect to that daemon, and have the same "fake" view of the file system's permissions/ownerships. (assuming the daemon and connecting program were started by the same user).

LIMITATIONS

Library versions

Every command executed within fakeroot needs to be linked to the same version of the C library as fakeroot itself.

open()/create()

fakeroot doesn't wrap open(), create(), etc. So, if user joost does either

```
touch foo
fakeroot
ls -al foo
```

or the other way around,

```
            fakeroot
            touch foo
            ls -al foo

            fakeroot has no way of knowing that in the first case, the owner
            of foo really should be joost while the second case it should
            have been root. For the Debian packaging, defaulting to giving
            all "unknown" files uid=gid=0, is always OK. The real way around
            this is to wrap open() and create(), but that creates other
            problems, as demonstrated by the libtricks package. This package
            wrapped many more functions, and tried to do a lot more than
            fakeroot . It turned out that a minor upgrade of libc (from one
            where the stat() function didn't use open() to one with a stat()
            function that did (in some cases) use open()), would cause unex-
            plainable segfaults (that is, the libc6 stat() called the
            wrapped open(), which would then call the libc6 stat(), etc).
            Fixing them wasn't all that easy, but once fixed, it was just a
            matter of time before another function started to use open(),
            never mind trying to port it to a different operating system.
            Thus I decided to keep the number of functions wrapped by fake-
            root as small as possible, to limit the likelihood of 'colli-
            sions'.

      GNU configure (and other such programs)
            fakeroot, in effect, is changing the way the system behaves.
            Programs that probe the system like GNU configure may get con-
            fused by this (or if they don't, they may stress fakeroot so
            much that fakeroot itself becomes confused). So, it's advisable
            not to run "configure" from within fakeroot. As configure should
            be called in the "debian/rules build" target, running
            "dpkg-buildpackage -rfakeroot" correctly takes care of this.

BUGS
      It doesn't wrap open(). This isn't bad by itself, but if a program does
      open("file", O_WRONLY, 000), writes to file "file", closes it, and then
      again tries to open to read the file, then that open fails, as the mode
      of the file will be 000. The bug is that if root does the same, open()
      will succeed, as the file permissions aren't checked at all for root. I
      choose not to wrap open(), as open() is used by many other functions in
      libc (also those that are already wrapped), thus creating loops (or
      possible future loops, when the implementation of various libc func-
      tions slightly change).

COPYING
      fakeroot is distributed under the GNU General Public License. (GPL 2.0
      or greater).

AUTHORS
      joost witteveen
            <joostje@debian.org>

      Clint Adams
            <schizo@debian.org>
```

```
       Timo Savola

MANUAL PAGE
       mostly  by  J.H.M.  Dassen <jdassen@debian.org> Rather a lot mods/addi-
       tions by joost and Clint.

SEE ALSO
       faked(1) dpkg-buildpackage(1), debuild(1) /usr/share/doc/fakeroot/DEBUG



Debian Project                   6 August 2004                      fakeroot(1)
```