Use GUIX to fix an upstream package issue

2023-12-25T15:41:52+01:00

I am using a Talos II PowerPC (ppc64le) machine as my daily computer. This poses some challenges every now and then as the support for that architecture is less ubiquitous than the *default* x86-64. It's the price to pay for having a completely open, documented machine.

On this system I run *archpower* distribution to get a linux kernel onto the machine, but I prefer to run GUIX on top of it for package management. I still need to run the native `pacman` package manager though, some packages are either not in GUIX yet (not so much of a problem in practice) or do not support ppc64le at all which is a bigger problem.

There's a third category, which I'd like to write about in this post. Packages that do support the architecture, but it's not a first class citizen. With this I mean that support is there, but the PowerPC package gets less attention and testing. This is somewhat inevitable as far less people use these machines rather than a *normal pc*, so it's likely they won't get as much eyes.

Depending on your distribution and package manager you can get stuck on a certain version for a package if there are limited options to upgrade or building the package locally is an effort out of your reach or time availability. Often the only option, for me anyways, is waiting for upstream to fix the situation. For this reason I still can't have packages that depend on rust in GUIX.

Because GUIX is basically a scheme library, you can use it to define how packages get into your machine and there is usually a better option to create a solution without having to figure out all the build details of the package itself.

One example of this is a recent ppc64le specific issue with OpenSSH.

The problem was the release (9.6p1) which was packaged for GUIX, and got eventually into my system on a `GUIX pull` command. However, the package build failed, apparently only on ppc64le machines due to this issue.

Because OpenSSH is used by many other packages, its failing build affected all those packages.

Here's what I did in GUIX to get to the new OpenSSH version which contained a number of security fixes which I wanted to have.

First, I created a package definition called `openssh-next` which takes the existing `openssh` package in GUIX and inherit from it in such a way that the fix for the problem outlined above will be included. In this case, take the commit just after the release from upstream.

```
;; Define openssh-next package which takes openssh from upstream which has the fix applied
;; See https://github.com/openssh/openssh-portable/commit/1036d77b34a5fa15e56f516b81b992800
(define-public openssh-next
  (let ((xcommit "1036d77b34a5fa15e56f516b81b9928006848cbd"))
    (package
      (inherit openssh)
      (name "openssh-next")
      (version "9.6p1-1")
      (native-inputs
       (list autoconf
             automake
             pkg-config))
      (source
       (origin
         (method git-fetch)
         (uri (git-reference
               (url "https://github.com/openssh/openssh-portable.git")
               (commit xcommit)))
         (file-name (git-file-name name version))
         (patches (search-patches "openssh-trust-guix-store-directory.patch"))
         (sha256
          (base32 "1sary1ig972l4zjvpzncf9whfp5ab8snff2fw9sy5a8pda5n2a7w")))))))
```

The crux in the above snippets is the `inherit` line which hides all the package definition complexity and the adapted `source` block which takes a specific git commit from the OpenSSH repository. Another way would be to add an extra `patch` line in the source block which contains just the upstream fix for ppc64.

With this new package definition, the new version can be installed, but all packages which depend on `openssh` are still using the original version. We want some way to go over everything that depends on `openssh` and replace its dependency with the new `openssh-next` package.

This is where GUIX can make your life easier. The GUIX api provides a couple of ways to do this. I built it up around the functions package-input-rewriting/spec and package-mapping.

The first takes a list of replacements, where each element of the list is a pair of a package spec and a procedure passed with a package to replace it with.

The second is a function to apply a function to a package to apply the defined replacement to a package. I wrapped both functions to make the call for the relevant packages a bit simpler.

```
;; Given a package spec, Replace input `old` with `new` for that package incl. its dependent
;; Return a procedure which takes the package as parameter
(define (package-input-replace old new)
  (package-input-rewriting/spec
   `((,old . ,(const (specification->package new))))))

;; Apply the input replace for openssh
;; pass each package which fails to build due to openssh as dependency
(define (openssh-fix package)
  ((package-mapping
    (package-input-replace "openssh" "openssh-next"))
   (specification->package package)))

;; Two examples on what to put in the manifest
(openssh-fix "gvfs")
(openssh-fix "remmina")
```

With this solution in place, the whole local package set builds again (takes a while though, as it often does with GUIX) and I can take advantage of the new OpenSSH release. There is a bit of 'keeping an eye on it' involved from this point on though. If upstream fixes the issue I want to take the above scheme code out of my manifest again and start using the upstream `openssh` package again instead of my `openssh-next` definition.