A Haskell Binding for OpenBSD Pledge

Björn Gohla

2023-09-16

Overview

EuroBSDCon 2023, Coimbra

- OpenBSD Pledge
- Haskell
- Design
- Code examples

Pledge

Restrict kernel API

```
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv){
  pledge ("stdio","");
  printf ("hello\n");
  pledge ("","");
  printf ("hello again\n"); // process gets terminated
}
```

Haskell Language

- first defined in the 1990s
- functional

```
(\x -> x + x)
```

```
(length . revert) [1,2,3]
```

- lazily evaluated

```
(\x -> x * x) (2 + 2)
```

- pure
- with a good type system

GHC

The Glasgow Haskell Compiler

- de facto standard compiler, REPL and RTS
- various backends and platforms
- m:n multithreading
- software transactional memory
- garbage collector

Haskell Ecosystem

- Hackage: central package repo
- Hoogle: search engine
- Cabal: package manager and build system
- Haskell Language Server

Hello World

```
module Main where

import System.IO (appendFile)

formGreeting :: String -> String
formGreeting n = "hello " <> n <> "!"

main :: IO ()
main = do
  putStrLn "what is your name?"
  name <- getLine
  appendFile "names.txt" $ name <> "\n"
  putStrLn $ formGreeting name
```

With Promises

```
module Main where

import System.IO (appendFile)

formGreeting :: String -> String
formGreeting n = "hello " <> n <> "!"

main :: IO ()
main = do
  putStrLn "what is your name?" -- stdio
  name <- getLine -- stdio
  appendFile "names.txt" $ name <> "\n" -- wpath, stdio
  putStrLn $ formGreeting name -- stdio
```

What to Pledge

```
module Main where

import System.IO (appendFile)

import System.OpenBSD.Pledge.Promise.Type
import System.OpenBSD.Pledge.Internal

formGreeting :: String -> String
formGreeting n = "hello " <> n <> "!"

main :: IO ()
main = do
  pledge _
  putStrLn "what is your name?" -- stdio
  pledge _
  name <- getLine -- stdio
  pledge _
  appendFile "names.txt" $ name <> "\n" -- wpath, stdio
  pledge _
  putStrLn $ formGreeting name -- stdio
  pledge $ fromList []
```

Solution

```
module Main where

import System.IO (appendFile)

import System.OpenBSD.Pledge.Promise.Type
import System.OpenBSD.Pledge.Internal

formGreeting :: String -> String
formGreeting n = "hello " <> n <> "!"

main :: IO ()
main = do
  pledge $ fromList [Stdio, Wpath]
  putStrLn "what is your name?"
  pledge $ fromList [Stdio, Wpath]
  name <- getLine
  pledge $ fromList [Stdio, Wpath]
  appendFile "names.txt" $ name <> "\n"
  pledge $ fromList [Stdio]
  putStrLn $ formGreeting name
  pledge $ fromList []
```

A Puzzle

```
do
  s <- getLine -- IO String
  putStrLn s -- String -> IO ()
```

How is this functional?

With Sugar

```
do
  s <- getLine -- IO String
  putStrLn s -- String -> IO ()
```

is actually

```
getLine >>= (\s -> putStrLn s)
```

Bind

```
getLine >>= (\s -> putStrLn s)
```

where

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

Label Actions

with promoted types

```
-- System.OpenBSD.MultiPledge
newtype Pledge (zs :: [Promise]) (ps :: [Promise]) a
  = Pledge { getAction :: IO a }
```

requires some explanation

The Explanation

DataKinds promotes data constructors to types

```
Stdio :: Promise
Promise :: *
'Stdio :: Promise

[Stdio, Inet] :: [Promise]
[Promise] :: *
'[ 'Stdio, 'Inet] :: [Promise]
```

For example

annotate base functions

```
-- System.Directory
getDirectoryContents :: FilePath -> IO [FilePath]
```

```
import qualified System.Directory           as D (getDirectoryContents)
```

```
getDirectoryContents :: FilePath -> Pledge zs '[ 'Rpath] [FilePath]
getDirectoryContents = Pledge . D.getDirectoryContents
```

A New Bind Operator

```
-- System.OpenBSD.MultiPledge
(>>=) :: forall zs ps m qs a b.
      ( MonadIO m, SingI zs, SingI ps, SingI qs
      )
   => Pledge (zs `Union` ps) qs m a
   -> (a -> Pledge zs ps m b)
   -> Pledge zs (ps `Union` qs) m b
```

best explained with a picture and some code

Caveats

- no multithreading
- no exec promises
- redundant pledge calls
- not portable (so far)

Resources

- Haskell Pledge
- Haskell.org
- Abadi et al., A core calculus of dependency

Thanks

Björn Gohla
https://mathstodon.xyz/@6d03
https://6d03.info