

Bound Threads

interfacing light-weight threads
with operating system threads

Part I — The Problem

Lightweight Threads

- GHC and Hugs don't use OS-supplied multithreading facilities
- They use just one OS thread for many Haskell threads.
- Superior Performance in GHC
- Simpler implementation in Hugs

Two Kinds of Threads

- OS Thread (also Native Thread):
 - created using an OS-supplied function
- Haskell Thread:
 - created by `forkIO`
 - or by calling Haskell from C

Blocking Foreign Calls

- During a foreign call, the Haskell world stands still
- OK for "unsafe" foreign imports
- NOT OK for "safe" foreign imports
 - not intended for thread synchronization

GHC's Threaded RTS

- `./configure --enable-threaded-rts`
- Haskell may be called from multiple threads
- No blocking: Run-Time System (RTS) creates a new OS thread when needed
- One Haskell Thread may be executed by different OS Threads during its lifetime

Thread-Local State

- OS threads can be uniquely identified (thread IDs)
- Libraries can request additional thread-local storage

Thread-Local State (continued)

- Thread-Local State is used by:
 - `errno`
 - OpenGL
 - Apple's "Carbon" and "Cocoa" GUI toolkits
 - Recursive Locks

The Problem

- These libraries ...
 - ... are incompatible with non-blocking foreign calls
 - ... can only be used from a single Haskell thread

Part II — The Solution

a.k.a. the Bound Threads Proposal

Bound Threads

- Every Haskell thread is either unbound, or bound to exactly one OS thread
- At most one Haskell thread may be bound to one OS thread
- `forkIO` forks a new unbound Haskell thread
- `forkOS` creates a Haskell thread that is bound to a new OS thread

foreign import

- A foreign call made by a bound Haskell thread is always executed by its bound OS thread
- A bound OS thread may not be used to execute foreign calls from another Haskell thread
- "Safe" foreign calls never cause other Haskell threads to block

foreign export

- A foreign call into Haskell is run by a Haskell thread bound to the OS thread that made the call

Implementation Independence

- We never say that lightweight threads have to exist (using OS threads for everything is OK)
- We only talk about foreign calls; Haskell code may be executed in any OS thread

errno

- errno is OS-thread-local
- errno is used throughout the libraries, so using bound threads here is overkill
- The RTS should manage errno for each Haskell thread
- make errno "Haskell-thread-local"

Primitives

- `forkIO :: IO () -> IO ThreadId`
- `forkOS` is a library function
- `forkProcess :: IO (Maybe ProcessID)`
- `isCurrentThreadBound :: IO Bool`

Library Functions

- `forkOS :: IO () -> IO ThreadId`
- `runInBoundThread :: IO a -> IO a`
 - creates a new bound thread only if the current thread isn't bound
 - waits for the action to complete

forkOS

```
foreign import ccall "wrapper" mkEntryPoint  
  :: IO () -> IO (FunPtr (IO ()))
```

```
foreign import ccall createOSThread  
  :: Ptr (OSThreadId) -> FunPtr (IO ())  
  -> IO CInt
```

```
forkOS action = do  
  mv <- newEmptyMVar mv  
  entry <- mkEntryPoint  
    (myThreadId >>= putMVar mv >> action)  
  alloca $ flip createOSThread entry  
  takeMVar mv
```

runInBoundThread

```
runInBoundThread :: IO a -> IO a
```

```
runInBoundThread action = do
  bound <- isCurrentThreadBound
  if bound
    then action
    else do
      mv <- newEmptyMVar
      forkOS (action >>= putMVar mv)
      takeMVar mv
```

Open Issues

- "forkOS" isn't a good name.
- Should "main" run in a bound thread?
- Should forkProcess be allowed from bound threads only?
- Non-GHC Implementations

Formal Specification

OS Threads (a.k.a. Native Threads)

Native thread	$t ::= N[S]$	
Native thread stack	$S ::= \epsilon$	Empty
	$H : S$	Executing Haskell
	$F^{si} a_{bt} : S$	Executing foreign code
	\bullet	Unknown
Safety indicator	$si ::= u$	Unsafe
	s	Safe

Formal Specification

Haskell Threads

Haskell thread	h	$::=$	$(a)_{bt}$	
Bound thread id	bt	$::=$	ϵ	Not bound
			N	Bound to native thread N
Haskell action	a	$::=$	$p \gg a$	Sequence
			RET	Return from a call into Haskell
Primitive action	p	$::=$	τ	Internal action
			$\text{forkIO } a$	Fork a thread
			$\text{forkOS } a$	Fork a native thread
			$F^{si} f$	Foreign call

Formal Specification

Haskell Threads

$$N[H : S]; (\tau \gg a)_{bt} \Rightarrow N[H : S]; (a)_{bt}$$

$$N[H : S]; (\text{forkIO } b \gg a)_{bt} \Rightarrow N[H : S]; (a)_{bt}, (b)_{\epsilon}$$

$$; (RET)_{\epsilon} \Rightarrow ;$$

Formal Specification

OS Threads

$(nothing) \Rightarrow N[H];$
where N is fresh

$(nothing) \Rightarrow N[\bullet];$
where N is fresh

$N[S]; \Rightarrow (nothing)$

Formal Specification

foreign import

$$\begin{aligned} N[H : S]; (F^{si} f \gg a)_N &\Rightarrow N[F^{si} a_N : H : S]; \\ N[H]; (F^{si} f \gg a)_\epsilon &\Rightarrow N[F^{si} a_\epsilon : H : S]; \end{aligned}$$

$$N[F^{si} a_{bt} : S]; \Rightarrow N[S]; a_{bt}$$

Formal Specification

foreign export

$$N[\bullet]; \Rightarrow N[H : \bullet]; (f \gg RET)_N$$

$$N[F^s a : S]; \Rightarrow N[H : F^s a : S]; (f \gg RET)_N$$

$$N[H : S]; (RET)_N \Rightarrow N[S];$$