

Linux (Kernel) Modules in with Haskell

By Thomas M. DuBuisson

Overview

- GHC & Library Alterations
- Module code
- Compiling and linking
- Engineering larger modules

Ways to Use Haskell with Linux

- Force Linus to rewrite / use Haskell for all of Linux
- Use GHC to build modules
 - Haskell code
 - Interpret Kernel macros
 - Static data structures
- Use the Linux build system to build modules and call exported Haskell functions (manual linking)

GHC Alterations

- Start with GHC-House
- Add a calling convention
- Strip out “bare metal” assumptions
 - respect the interrupt flag
 - Don't manage system memory
- No common area for uninitialized globals
(-fno-common)
- Add a 'shutdown' routine for the RTS

Module Code (c shim)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/slab.h>
#include "Kernel/memory.h"
```

```
MODULE_LICENSE("Dual BSD/GPL");
```

```
static int hello_init(void)
{
    return hello();
}
static void hello_exit(void)
{
    goodbye();
}
module_init(hello_init);
module_exit(hello_exit);
```

Large Modules (why C)

- Global structs

```
struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
...

```

- To get macros (GHC CPP can't handle Linux)

```
module_init(ricoh_mmc_drv_init_c);
module_exit(ricoh_mmc_drv_exit_c);

MODULE_AUTHOR(...)
MODULE_DESCRIPTION(...)
MODULE_LICENSE("GPL");
```

Module Code: FFI

- Add a GHC FFI for regparm3

```
foreign import regparm3 safe "pci_unregister_driver"  
  pciUnregisterDriver :: PCIDriver -> IO ()
```

```
foreign export regparm3 ricoh_module_init :: IO ()
```

- A shim for RTS calls that remain regparm0

```
void * __attribute__((regparm(0))) malloc(size_t s)  
{  
  return kmalloc(s, ALLOC_MODE);  
}
```

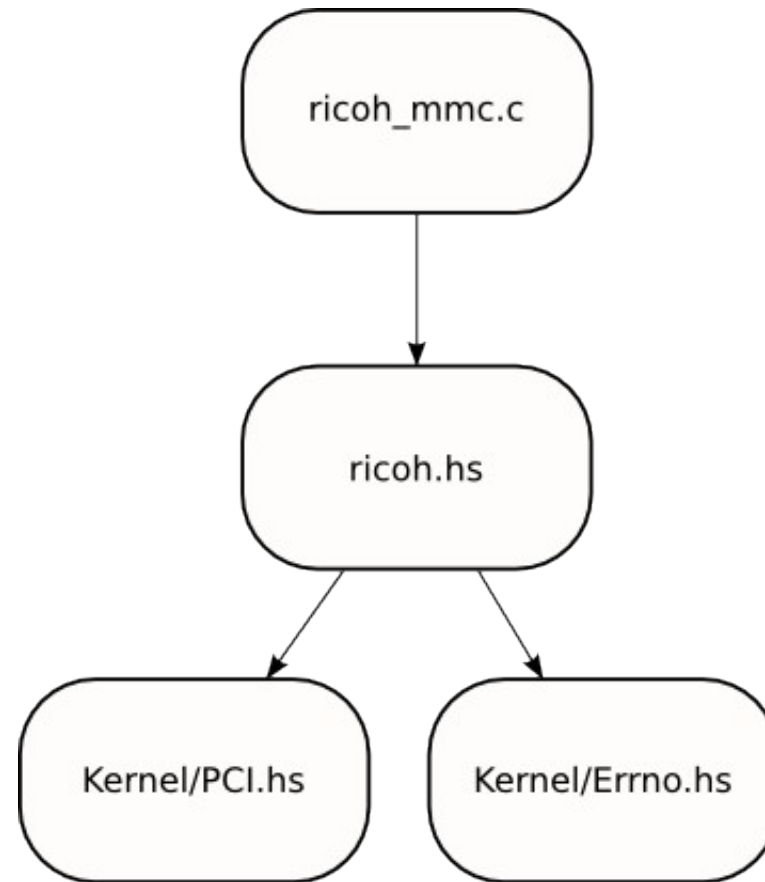
Module Code (Haskell)

```
hello :: IO CInt
hello = newCString "hello" >>= printk >> return 0

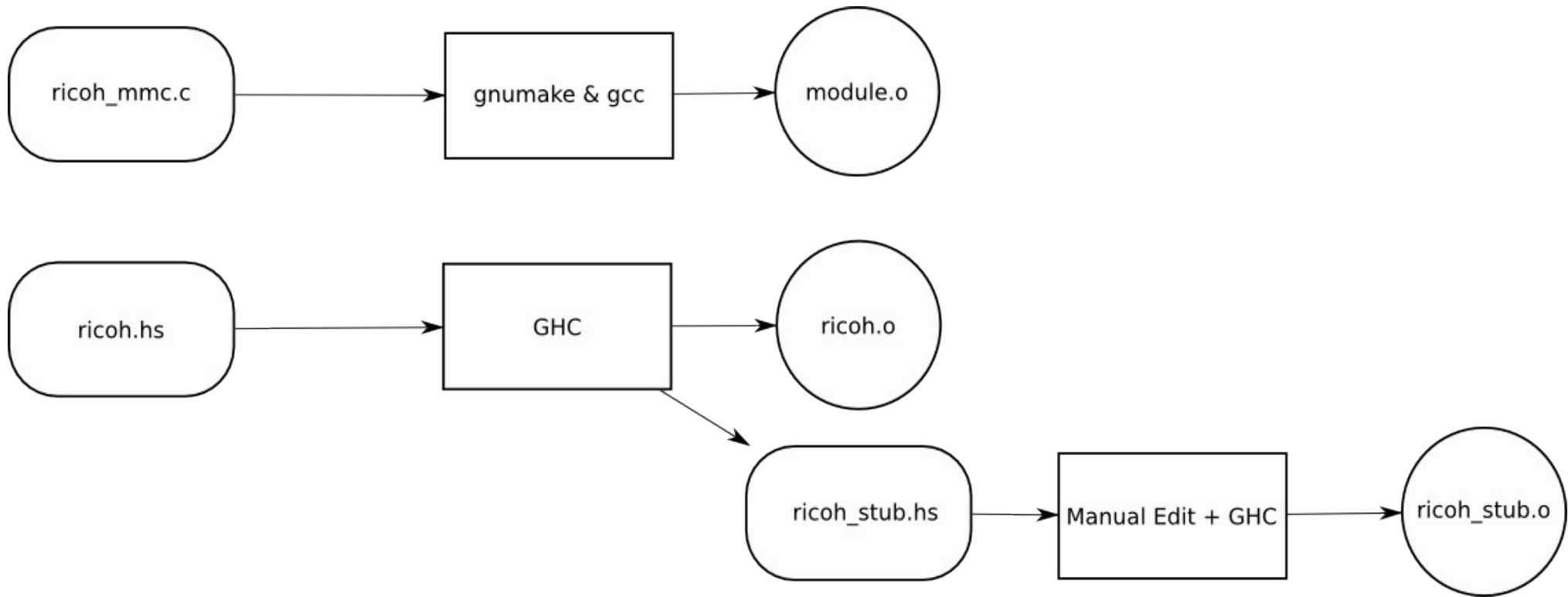
foreign export regparm3 hello :: IO CInt
foreign import regparm3 unsafe printk :: CString -> IO Int
```

```
ricohMMCEnable :: PCIDevice -> IO CInt
ricohMMCEnable dev
  | device dev == pciDeviceIdRicohRL5C476 = do
    writeEnable <- read 0x8E
    write 0x8E 0xAA
    ...
  | otherwise = do
    writeEnable <- read 0xCA
    disable <- read 0xCB
    ...
where enabledMessage = printk "Controller is now re-enabled.\n"
      write = pci_write dev
      read = pci_read dev
```

General Idea: Call Graph



General Idea: Compiling



Build Process

- Compile a .c file that uses Linux module registration macros
- Generate the *_stub.c file
- **Modify the stub to start and stop the RTS**
- Compile the stub using GHC
- Link everything, creating your .ko file.

Larger Modules

- There is no “Kernel API” that can be translated for a Haskell / C Binding.
- hsc2hs is full of features but assumes stdlib
- c2hs has fewer features but works

```
dev :: PCIDevice -> IO Device
dev = {#get &pci_dev_t->dev#}
```

Large Modules

- There is no “Kernel API” that can be translated for a Haskell / C Binding.
- hsc2hs is full of features but assumes stdlib
- c2hs has fewer features but works

```
dev :: PCIDevice -> IO Device  
dev = {#get &pci_dev_t->dev#}
```

```
#c  
device_t *c_dev(pci_dev_t *pdev)  
{  
    return (&pdev->dev);  
}  
#endc  
  
foreign import regparm3 unsafe "c_dev"  
dev :: PCIDevice -> Device
```

Gotchas!

- No explicit freeing of nurseries
- Memory allocation needs context

```
void *kmalloc(size_t n, gfp_t flags)
```

- Interrupts are an open question
- Concurrency needs work

Questions?

Appendix (Build Process)

Generate the .h and .hs files from chs:

```
c2hs -C-D__KERNEL__ -C-Wall -C-Wundef -C-Wstrict-prototypes -C-Wno-trigraphs -C-fno-strict-aliasing -C-fno-common -C-Werror-implicit-function-declaration -C-fno-delete-null-pointer-checks -C-Os -C-m32 -C-msoft-float -C-mregparm=3 -C-freg-struct-return -C-mpreferred-stack-boundary=2 -C-march=i586 -C-mtune=generic -C-Wa,-mtune=generic32 -C-ffreestanding -C-DCONFIG_AS_CFI=1 -C-DCONFIG_AS_CFI_SIGNAL_FRAME=1 -C-pipe -C-Wno-sign-compare -C-fno-asynchronous-unwind-tables -C-mno-sse -C-mno-mmx -C-mno-sse2 -C-mno-3dnow -C-I/usr/src/kernels/2.6.30.8-64.fc11.i586/arch/x86/include/asm/mach-generic -C-I/usr/src/kernels/2.6.30.8-64.fc11.i586/arch/x86/include/asm/mach-default -C-Wframe-larger-than=1024 -C-fno-stack-protector -C-fno-omit-frame-pointer -C-fno-optimize-sibling-calls -C-g -C-pg -C-Wdeclaration-after-statement -C-Wno-pointer-sign -C-fwrapv -C-DCONFIG_X86_32 -C-fno-dwarf2-cfi-asm -C-nostdlib -C-I$a -C-I$k -C-DCONFIG_PARAVIRT PCI.chs
```

Make sure the generated **.h files are #included in the .c file** being build by the linux kernel

```
make -C /usr/src/kernels/2.6.30.8-64.fc11.i586 M=`pwd` modules
```

```
$HOUSEGHC/compiler/stage1/ghc-6.8.2 -B$HOUSEGHC/ghc-6.8.2 ricoh.hs -c -fvia-c
```

Add startupHaskell and hs_exit_nowait to the **_stub** file init and exit routines.

```
ld -r -m elf_i386 -o module.ko ricoh_stub.o ricoh.o module.mod.o ricoh_mmc.o  
Kernel/PCI.o Kernel/Errno.o ../libs/libHSbase-3.0.1.0.a ../libs/libHSrts.a ../libs/libtiny_c.a  
../libs/libtiny_gcc.a ../libs/libtiny_gmp.a ../libs/libtiny_m.a ../libs/libcbits.a
```